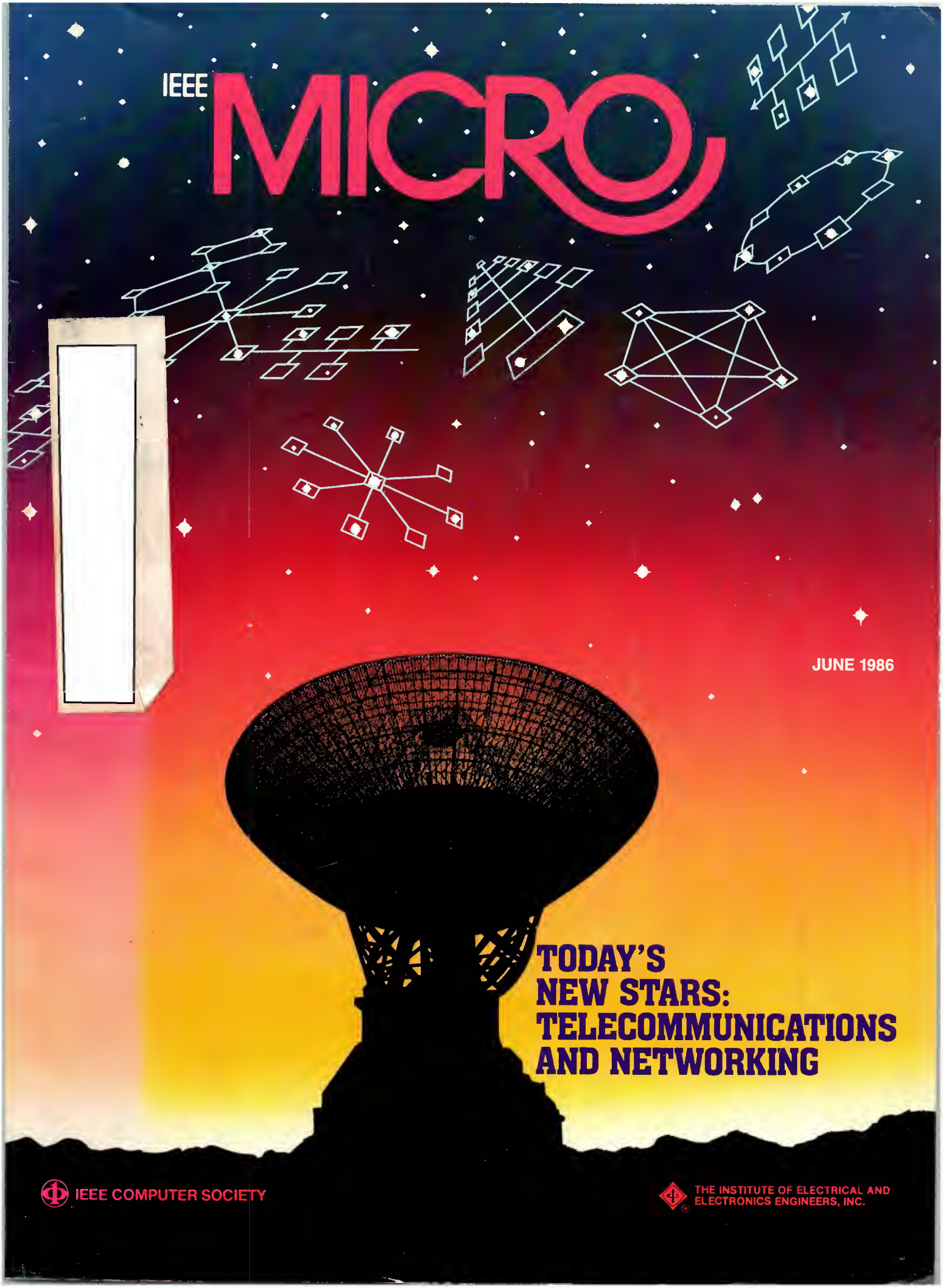
IEEE MICRO



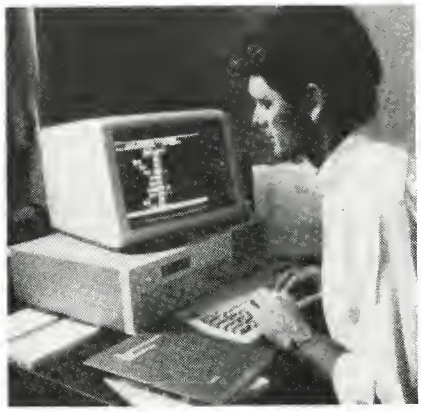
JUNE 1986

TODAY'S NEW STARS: TELECOMMUNICATIONS AND NETWORKING

IEEE COMPUTER SOCIETY

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

Announcing new PC SIMSCRIPT II.5 . . . with animation



SIMSCRIPT II.5 with animation now on personal computers

free trial--see how SIMSCRIPT II.5 helps you build a realistic model

the complete SIMSCRIPT II.5 on a PC

SIMSCRIPT II.5 for personal computers is the same popular simulation language that is now widely used on mainframes.

You can now build realistic models of military, manufacturing, communications, logistics, transportation or other systems on your PC.

PC SIMSCRIPT includes a new programming environment that makes it easy for you to develop, verify, modify, and enhance simulation models on a personal computer.

natural method of modelling

When building a model in SIMSCRIPT, you describe the simulated system as consisting of certain types of entities: perhaps workers, machines and jobs in a simulated factory; or flights and airports in a simulated air transport system; or jobs, processors, channels, and I/O devices in a simulated computer system.

For each type of entity you give names to the attributes that characterize it.

You also name the sets an entity type may belong to, and the sets it may own.

Since your model is English-like, with names that you choose, it reads like a description of the simulated system. The model can be read and verified by non-programmers who understand the system under study.

This makes your model development, validation and evolutionary changes much easier.

large models on your PC

Your model and data are not limited by the size of the PC. SIMSCRIPT is the only simulation tool that automatically makes use of the hard disk as a memory extension.

reduced cost

SIMSCRIPT II.5® is a well established, standardized, and widely used language with proven software support.

Experience has shown that SIMSCRIPT II.5 reduces simulation programming time and cost severalfold compared to other simulation techniques.

animated and graphical results

With PC SIMSCRIPT II.5® you build models that can show an animated picture of the system under study. Observing the simulation improves understanding of the system and builds confidence in the model.

Because you see the operation of the simulated system and can easily try alternatives, the time and cost of system analysis are sharply reduced.

computers with SIMSCRIPT II.5

1. IBM Personal Computer AT, XT, PC or compatible, with a hard disk.

2. Most Mainframe computer types including IBM, CDC, VAX, Univac, Prime, Gould, Data General and Honeywell.

SIMSCRIPT II.5 and PC SIMSCRIPT II.5 are registered trademarks and service marks of CACI, INC.-FEDERAL

free trial

The free trial package contains everything you need to try SIMSCRIPT II.5 on your own computer.

We send you PC or Mainframe SIMSCRIPT II.5, installation instructions, sample models, and a complete set of documentation. You can build your own model or modify one of ours. No cost or obligation.

special offer free training

For a limited time we will also include free training. **Space is limited so act now to avoid disappointment.**

Call Rick Crawford at (619) 457-9681 to reserve your place.

free trial-learn the reasons for the broad and growing popularity of SIMSCRIPT II.5—no cost or obligation

special offer-return the coupon today and we will include one free course enrollment worth $850

Name

Organization

Address

City State Zip

Telephone

Computer Operating System

Return to: IEEE MICRO

CACI

3344 North Torrey Pines Court
La Jolla, California 92037

Or, better yet,
call Rick Crawford at (619) 457-9681

IEEE MICRO

Volume 6 Number 3 (ISSN 0272-1732) June 1986

On the cover
LAN architectures mimic constellations in an evening sky over a silhouetted satellite receiving dish.

Cover: Jay Simpson and Larry Keiser
Article design/production: Alexander Torres
Typesetting: Fagen Graphics

In the next issue:

Multiprocessing

Feature Articles

Special Features

MicroStandards Special Feature

Departments

Circulation: *IEEE Micro* (ISSN 0272-1732) is published bimonthly by the IEEE Computer Society: IEEE Headquarters, 345 East 47th St., New York, NY 10017; IEEE Computer Society West Coast Office, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578. Annual subscription: $12.00 in addition to IEEE Computer Society or any other IEEE society member dues. Nonmember prices: available on request. Single-copy prices: members $7.50; nonmembers $15.00. This journal is also available in microfiche form.

Undelivered copies: Send to 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578.

Postmaster: Send address changes to *IEEE Micro*, IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854. Second class postage is paid at New York, NY, and at additional mailing offices.

From the Editor-in-Chief

Since the April issue, I have had the opportunity to attend two major conferences—Eurocon in Paris and Electro in Boston. While the technical content of both professional programs was excellent, the logistics varied greatly. The papers were presented in parallel sessions at both shows, and the topics were very similiar in many instances. Since Eurocon does not have an exhibit floor, attendance was much smaller than Electro. Eurocon was housed in the cavernous Palais de Congress, which had another major conference and several workshops proceeding simultaneously. There was still plenty of room left over. There was a huge hotel nearby.

In contrast, Electro and MiniMicro were again separated by a lengthy bus or taxi ride. This year, the usually overcrowded Hynes Auditorium in downtown Boston was in the process of being rebuilt, so a bus or cab ride was required to get to either the show floor or the Professional Program Sessions. I spent a lot of time on the bus in Boston. Perhaps when the Hynes Auditorium is completed, the situation will be relieved.

One session at MiniMicro was truly unusual. The session was "Bus Wars," chaired by Bob Stewart, *IEEE Micro*'s Standards Column editor for the past four years. Having completed two consecutive two-year terms on *Micro*, Bob selected his Electro overview paper to be his "swan song" to *Micro* readers.

Intel, Motorola, and National presented interesting and excellent papers, but Bob's opening overview was a showstopper. It was a comic strip. When I first saw it in print, I thought it was humorous whimsy, into which Bob occasionally indulges. Upon more careful inspection, I found that this was a historic (and humorous) accounting of what has happened to bus standards over the past ten years. While I am sure that some readers will be miffed at a comic strip appearing in the pages of a journal as august as *IEEE Micro*, I think it will prove interesting and informative to a large number of readers. I have asked Bob to write a brief forward to the strip, since not all of our readers live in Silicon Valley. Bob's final column will appear in the next issue.

I think it is proper to publicly thank Dr. Stewart for his efforts. While he was (and remains) sometimes controversial, there is no question that he expended a tremendous effort on behalf of bus standards, *IEEE Micro,* the IEEE Computer Society, and our profession in general. Thanks, Bob!

At both Eurocon and Electro I had the opportunity to discuss some technical issues, ways to increase *IEEE Micro*'s circulation here and abroad, and business in general with both people I knew and new acquaintances. Overall, the Europeans seemed more upbeat on the computer and semiconductor economy than the Americans were. Here in the US, I got the distinct impression of a cautious "wait-and-see" attitude.

In the mailbag this month were 42 reader responses. As usual, the articles were well received, but as frequently occurs, MicroLaw garnered the most favorable responses. J.A.A. of Fairfax, VA, went so far as to confess that he is really a software type and subscribes to *IEEE Micro* for Dick Stern's MicroLaw. J.A.A. does allow that he occasionally reads an article on hardware or systems.

D.P. (Hayward, CA) missed Bob Stewart in April.

D.C.H. (Binghamton, NY) wants more on video devices, fault-tolerant computing, and EPLDs.

L.E. (Stoughton, WI) likes DSP (wait until December, L.E.)

E.L. (Nedlands, West Australia, Australia) would like to see bus architectures for multiprocessing.

S.J. (Austin, TX) wants to see multiprocessing hardware/software tradeoffs.

O.G. (Istanbul, Turkey) asks for more detailed hardware and software information on 32-bit MPUs.

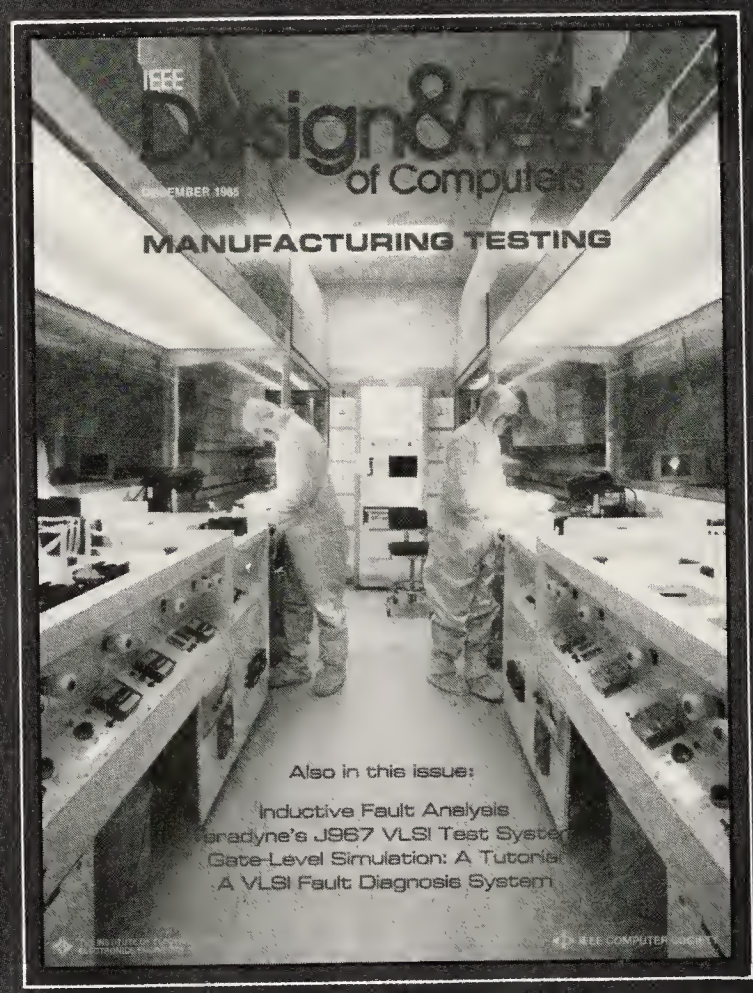
B.A.C. (Appleton, WI) would like to see keywords for correspondence when technical subjects are discussed.

A final note. One response that I received had a strange stamp on the front. The stamp is legitimate postage. It depicts a devil's hand coming forth from the United Nations building and pressing down on the earth. The arm is labeled "VETO" and the fingers are labeled "U.K.," "U.S.A," "U.S.S.R.," "F.R.C." (France), and "CHN" (China), representing the five permanent members of the U.N. security council. In full color, a sword is bloodily cutting off the hand. All politics aside, I feel that this is unfortunate and sad. I usually enjoy looking at stamps from countries all over the world.

Have a safe summer.

Jim Farrell

Designed & Tested by Professionals, for Professionals.



IEEE Design and Test of Computers is a technical magazine written for and by professionals in the design and test field. It is the medium by which colleagues share knowledge and help each other in the pursuit of excellence. Its articles are firsthand accounts by practicing experts about the methods for designing electronic equipment. You learn not just the theory but the empirical results of proven techniques such as built-in self-test strategies, expert systems for VLSI design, exploratory data analysis, and operational life testing techniques that you can apply right away.

D&T gives you perspective, techniques, and data to make your designs test-perfect. The magazine has been designed and tested by its readers. It works. And it will help you work better. Subscribe today.

☐ Computer Society or any other IEEE Society half-year member rate: $7 (three issues—Aug., Oct., Dec.)

Membership Number _____

☐ Sister Society rate: $25/year (six issues)

If you are a member of an affiliate society such as ACM, ASME, DPMA, SAE, SIAM, ASQC, or any other professional technical society, you are eligible for this special rate.

☐ Bill me
☐ Check enclosed
☐ Visa
☐ MasterCard
☐ American Express

Charge Card Number

Mo. | Yr.

Exp. Date

Signature

Name

Address

City State Zip

Return to: IEEE Computer Society
10662 Los Vaqueros Circle,
Los Alamitos, CA 90720-2578

MICROJUN

To the Editor

Von Neumann versus microcoding

To the Editor:

Your two recent letters on "von Neumann versus microcoding" were rather puzzling.[1,2]

My background in the subject is marginal: teaching EE courses on bit-slice (grad) and microcoded hardware (undergrad), assembly-language programming on several machines, interfacing, nine years in the ACM microprogramming SIG, and so on. However, several facets of the argument initially confused me.

Some examples may serve to illustrate the problem.

First, the IBM 360 was cited as a "classical von Neumann computer" by the "von Neumann cannot be microcoded" authors.[2] However, a microcoded APL implementation on a 360 was mentioned in the 1982 second edition of their first reference;[3] moreover, only the 360 Model 75 appears to have been built with hardwired control logic, while the other models (20, 25, 30, 40, 50, 60, 85) were microcoded.[4]

The Apple seems to be a von Neumann machine if you apply a Turing test (what you can observe from a terminal). If you open the box or read the schematic, it is still von Neumann. However, the silicon of the 6502 CPU chip appears microcoded. Most of the microprocessor chips of this generation are internally microprogrammed.[5]

The Commodore 64 also uses the 6502. The disk drive has its own CPU, so technically the 64 is not von Neumann; however, a Turing test cannot distinguish this second CPU from hardwired logic.

The Data General Nova has different instruction formats, all on a 16-bit word, and most of which may be viewed as having a 1-byte opcode followed by an address.[6] However, the arithmetic instruction specifies the two registers and the operation in the first byte, and "reconfiguration" options (shift, skip conditions, and "discard result" option) in the second byte. If "one of the most popular [computers] in the world" were not von Neumann, many more people would have discussed it in the literature.[7] My limited experience in hardware repair of Novas suggests that the earlier models use discrete logic for implementation, rather than microcode. The MicroNova chip is probably microcoded.

The Data General Eclipse, with substantially the same instruction set as the Nova, is microcoded. An optional writable control store lets you write your own microcode (e.g., a floating-point, 256-long complex FFT in one instruction).

The problem here and in the two earlier letters seems to be one of definition. A microprogram is defined by the ANSI standard[8] and ISO as "a sequence of elementary instructions that correspond to a computer operation, that is maintained in special storage, and whose execution is initiated by the introduction of a computer instruction into an instruction register of a computer." A von Neumann machine has been described[9] as having four characteristics:

(1) data and instructions share the same memory,
(2) memory is one dimenisonal (linear),
(3) instructions and data are indistinguishable, allowing self-modifying code, and
(4) data has no inherent meaning.

The microprogrammed machine by definition has a separate memory for microcode instructions, and therefore cannot be "von Neumann" by ANSI definition.[8] However, a Turing test usually cannot reveal the presence of fixed microprogramming.

To reduce future confusion, I suggest that future writers distinguish more clearly between microprogramming in instruction set (e.g., Nova), architecture (by Turing test), and implementation (the invisible hardware).

All company and machine names are trademarks.

James D. Marr
University of Alabama in Huntsville

References

1. T. Cain et al., "Review referee states concerns," *IEEE Micro*, Vol. 5, No. 5, Oct. 1985, pp. 4, 90-91.
2. S. Kartashev and S. Kartashev, "Authors' reply," *IEEE Micro*, Vol. 5, No. 5, pp. 91-93.
3. D. P. Siewiorek, C. G. Bell, A. Newell, *Computer Structures: Principles and Examples*, McGraw-Hill, New York, 1982.
4. S. S. Husson, *Microprogramming: Principles and Practices*, Prentice-Hall, Englewood Cliffs, N.J., 1970.
5. D. R. McGlynn, *Microprocessors: Technology, Architecture, and Applications*, Wiley-Interscience, New York, 1976.
6. *How to Use the Nova Computers*, Data General Corporation, Southboro, Mass., 1972.
7. A. Osborne and G. Kane, *Osborne 16-Bit Microprocessor Handbook*, Osborne/McGraw-Hill, Berkeley, Calif., 1981.
8. *American National Dictionary for Information Processing*, American National Standards Committee X3, Computers and Information Processing, Washington, DC, 1977.
9. G. J. Myers, *Advances in Computer Architecture*, Wiley-Interscience, New York, 1982.

Guest Editor's Introduction

1986: The Year of Networking

Arthur R. Miller
Motorola, Inc.

The message in 1986 from the users and potential users of both local area and wide area networks is loud and clear: Users want multivendor, interoperable networking systems. Many users will continue to purchase systems that they need now, even though they do not meet this requirement; but, given a choice, the users prefer the multivendor solution to a proprietary solution. The issues, as always, are long-term support and cost. The capability for providing such systems cost-effectively is being implemented in silicon and software. The mechanisms for developing interoperable products are standards groups, user groups, and testing.

As a relative newcomer to communications standards activities within IEEE 802, ANSI, and CCITT, I am impressed by the ability of a committee to produce networking standards that are robust and can be implemented to solve real-world problems in a cost-effective way. Academics no longer, if they ever really did, solely make up the committees. Major users of networks, and the vendors trying to meet their requirements, participate in force. Academic representatives, for the most part, immediately work on those issues related to implementing networks: testing, management, and optimization of performance. And, even more significant, all of these participants work with a sense of urgency.

Fast processors, inexpensive memory, and applications software fuel the spread of computing nodes that can be tied together. In some cases the costs of tying these nodes together limit the ability to apply computing solutions to some problems. Cheaper networking solutions may well accelerate the application of distributed computing. Since cheaper solutions require volume sales in our industry, vendors look for clear requirements from users—and they are receiving them in spades.

Networking user groups and closely related organizations of vendors have given us new sets of acronyms to learn: MAP (Manufacturing Automation Protocol), TOP (Technical and Office Protocols), X.OPEN (European ISO vendors), SPAG (Japanese ISO vendors), and COS (US ISO vendors). The relationships between the various country groups, user groups, and standards bodies are often confused and can reflect some parochialism and a resultant lack of synergy. Still, vendors are being driven to provide communications solutions by large, international systems users and by the user groups, which often reflect a significant share of their membership in vendor participants. This open participation of multiple vendors and multiple users seems to be working to produce multivendor systems implementations. It seems as if the open participation has also decreased the product development cycle for many vendors, especially during the product specification phase.

As a result of the efforts of standards bodies and user groups, issues of assuring interoperable systems now top the list of users and vendors alike. Most of the effort has been directed through MAP and the IEEE 802.4 committee working with the National Bureau of Standards, the GM MAP Task Force, and the Industrial Technology Institute. Relationships of tests to apply, the environment to apply them in, and the way to state the results are now being actively addressed. It seems likely that, at least in the case of the MAP scenario, a set of tests for a complete seven-layer ISO implementation will be ready by the end of 1986. These efforts are being leveraged by other communications scenarios, such as TOP, to define and implement testing for their distinct features. It may be possible someday to find an equivalent to the United Laboratories "UL" label on a communications *solution* from a number of vendors—and it will plug in and work!!

Articles in this issue address some of the specifics of communications solutions for standards now being implemented. The measure of connectivity now able to be implemented by standards is awesome (Yes, I do have teenagers!) and, of course, totally "application depen-

0272-1732/86/0600-0006$01.00 © 1986 IEEE

dent." I show a relatively simple representation of that connectivity in Figure 1. The drawing may appear to be confusing to the communications neophyte, but I hope it offers some insight into possible use of these protocol standards. Figure 1 also shows that there is considerable ground yet to cover. I hope that the fertile field of communications can be the theme for another *IEEE Micro* in the near future.

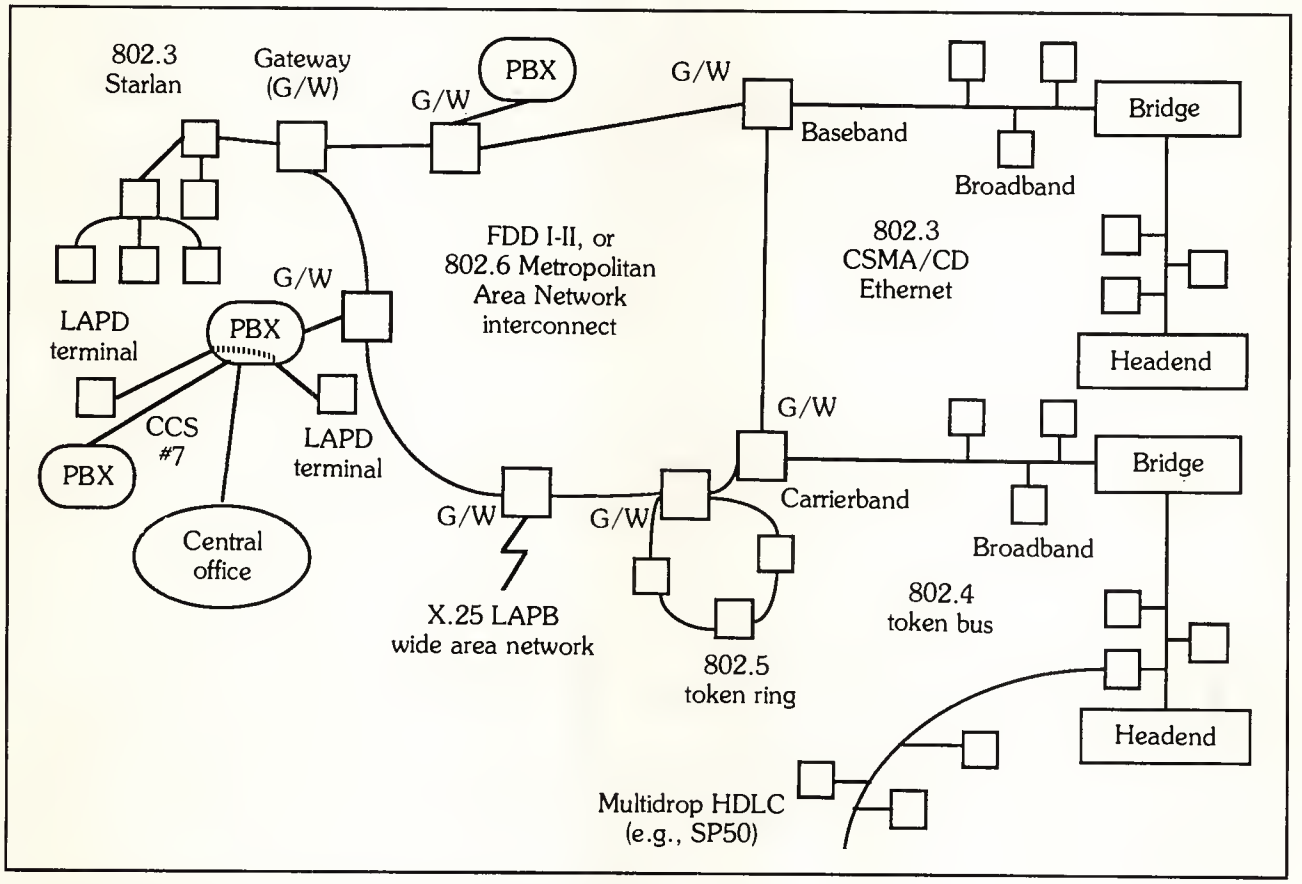


Figure 1. System interconnect environment—standards.

Arthur R. Miller is the business development manager for the Motorola 68000 microprocessor operation in Austin, Texas, where he is responsible for VLSI communications products. Prior to joining Motorola, he worked in the Digital Systems Group of Texas Instruments, doing computer systems product management. Miller received a BS in mathematics from the US Air Force Academy, an MSEE from MIT, and an MBA from St. Edwards University. He is a member of the MAP/TOP User Group and is chairman of the TOP Physical Access Subcommittee.

Miller can be reached at Motorola, Inc., Mail Stop OE-33, 6501 William Cannon Drive West, Austin, TX 78735-8598.

High-Performance Networks: A Focus on the Fiber Distributed Data Interface (FDDI) Standard

Sunil P. Joshi
Advanced Micro Devices

A new standard provides a high-performance LAN solution for the quick transfer of large amounts of data.

As the computing power of computers and the storage capacity of disk media increase with each passing day, there is increasing demand on communication links to transfer large amounts of data with minimum delay. The emerging Fiber Distributed Data Interface (FDDI) is a standard that provides a high-performance local area network solution for such high-end applications.

The FDDI standard is being defined by the American National Standards Institute committee X3T9.5, which is responsible for high-speed local area networks. FDDI is a fiber-optic token-ring network operating at a data rate of 100 megabits per second. The FDDI protocol has used the IEEE 802.5 token-ring protocol[1] as a starting point and enhanced it to enable cost-effective, high-speed implementation. The Institute of Electrical and Electronic Engineers and ANSI X3T9.5 work closely to prevent overlaps in activities. IEEE is defining standards at data rates below 50M bps, while the X3T9.5 committee is looking at speeds above that.

The initial FDDI specifications, referred to as the FDDI-1 standard, are better suited for data traffic than voice traffic. Those aspects of FDDI-1 that affect hardware implementation are already frozen as a standard. As an enhancement to FDDI-1 the committee has just begun work on a new definition called FDDI-2, which will provide a better interface for both data and voice traffic.

The FDDI-1 specification consists of four documents specifying the data-link layer and physical layer of the seven-layer Open Systems Interconnect model (see Figure 1).[2] The Physical Media Dependent (PMD) document deals with the fiber-optic cable, connectors, and jitter specifications at the electro-optic interface. The Physical Layer (PHY) document specifies the clock recovery and encoding mechanism. The Media Access Control (MAC) document specifies the token-passing protocol, and the Station Management (SMT) document deals with network management issues.[3]

Why fiber optics?

FDDI decided to use fiber optics as a transmission medium for a variety of reasons:

- *Bandwidth.* Fiber cable provides a very high capacity for carrying information. The data rate can be in the range of several hundred megabits per second.
- *Attenuation.* Fiber provides low attenuation resulting in efficient communication over several kilometers without repeaters.
- *Noise susceptibility.* Fiber cables transmit information as light and neither generate nor are affected by electromagnetic interference.
- *Security.* Since it is not easy to tap a fiber-optic cable without interrupting communication, fiber is more secure from malicious interception.
- *Cost.* The cost of fiber-optic cable has fallen considerably over the last two years, and fiber is projected to become cheaper than coaxial cable in a year or so. Since fiber is lightweight and thinner than coaxial cable, it is easier to pull through overcrowded ducts, which results in lower installation costs. In fact, the additional cost of putting a second fiber in the same cable is negligible; hence, FDDI has

0272-1732/86/0600-0008$01.00 © 1986 IEEE

opted for a duplex fiber cable and provided a high level of fault tolerance through this added redundancy.

In the past the cost of the laser diodes or avalanche photo diodes to interface with the fiber has been high. However, technological advances have provided low-cost LEDs and pin-diodes that can operate at the 100M-bps rate specified by FDDI.

Key application areas

The requirements and characteristics of a network vary, depending on the applications they try to support. Broadly speaking, one can identify four major application areas: the office floor, the computer room, the factory floor, and campus interconnections. Figure 2 illustrates the way a variety of applications could coexist in a typical company environment.

Office floor. A typical office environment is characterized by the proliferation of word processors, desktop personal computers, facsimile machines, terminals, and

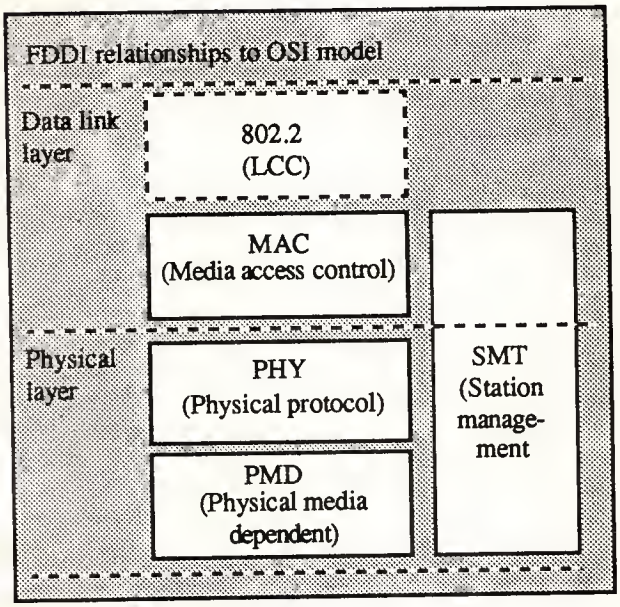


Figure 1. Scope of FDDI specifications.

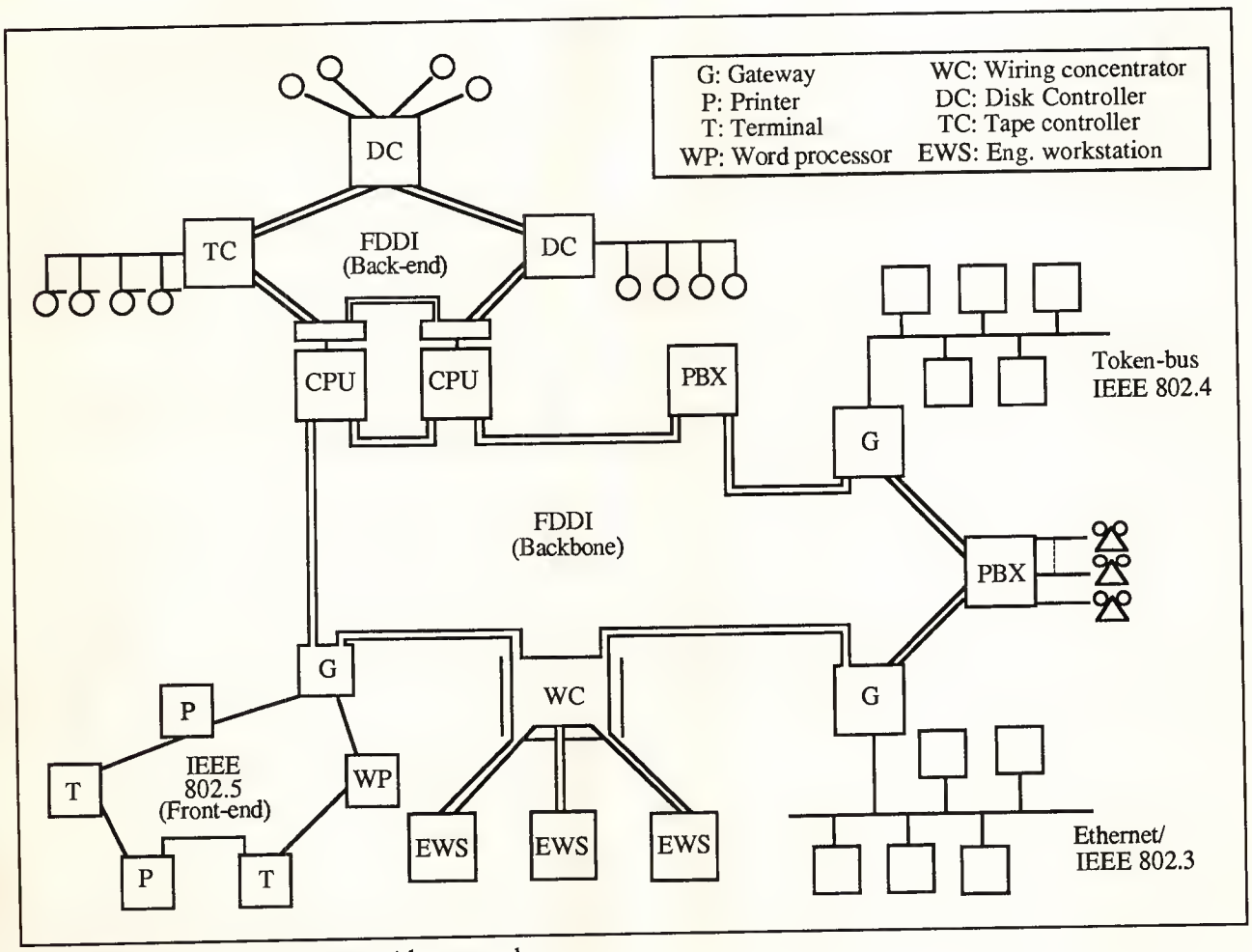


Figure 2. A view of a company-wide network.

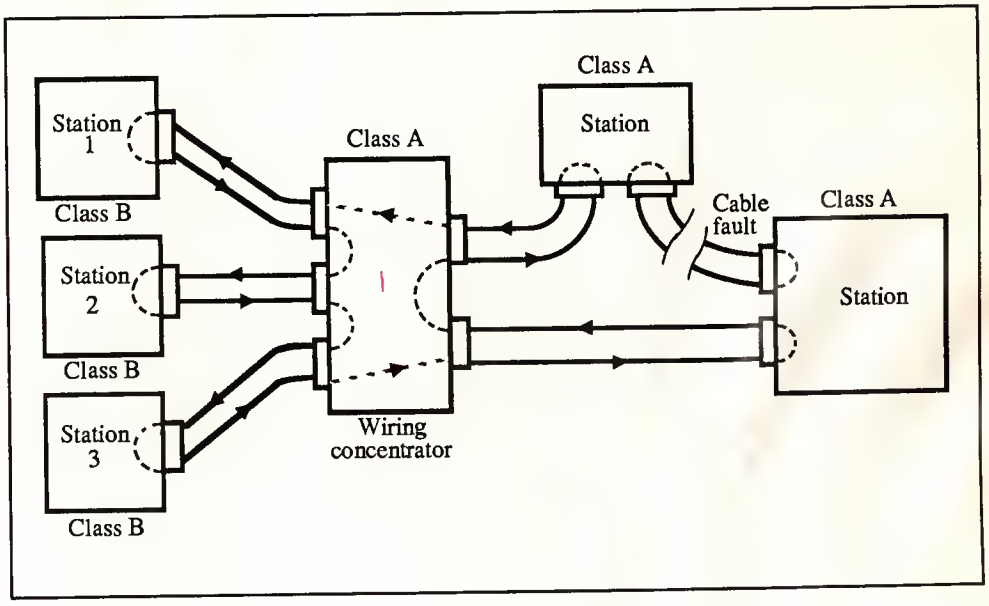


Figure 4. Ring reconfiguration.

Ring configuration. The most likely faults in a network are caused by either failed components or broken cables. Even if a connection is not fully broken, there may be a substantial degradation, which shows up as an increase in the bit-error rate. Figure 4 shows a ring reconfiguring its data paths when a link (or a pair of links in a cable) in the dual ring becomes inoperative. The stations sense the breakage and use the appropriate paths on the secondary ring to keep the network running. In FDDI this reconfiguration happens automatically, within a few milliseconds. FDDI defines a station management interface for this eventuality. When a broken ring is restored, the station management handshake will allow the ring to go back to its original state.

The effect of a second failure in the dual ring is interesting too. Figure 5 shows how a network can split into two smaller independent networks, which both remain operative internally.

If there is a cable fault in the cable going to a Class B station, this station is cut off from the network, and the WC provides a bypass for the node.

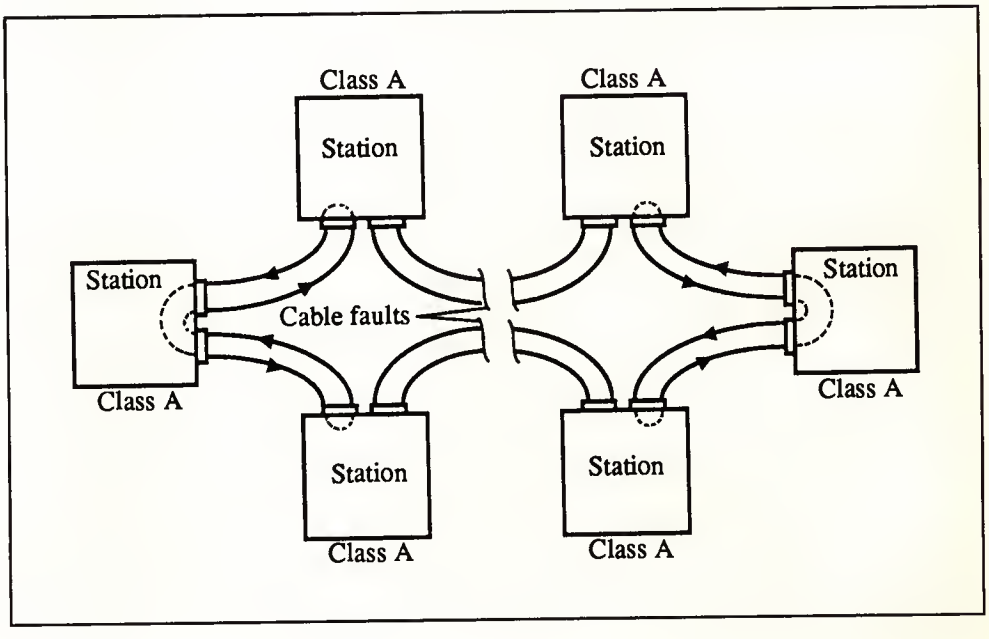


Figure 5. A ring with two cable faults.

Optical bypass. Situations in which stations connected to the ring lose electrical power or are turned off can be taken care of by providing optical bypasses. A bypass provides a path for the light to pass right past a node, using an electrically operated relay; when power is lost, a mirror directs the light through an alternative path.

A bypass can be provided in either Class A or Class B stations. Most Class A stations that include WCs will probably implement bypasses to get this additional level of fault tolerance. A Class A node without an optical bypass will appear as if all of its links are broken when powered off. Optical bypasses (or even electrical bypasses) may be used in WCs at their interface to Class B stations to take care of situations where the Class B stations may fail.

Encoding scheme. Several of the low-speed standards, including the IEEE standards, use Manchester encoding for baseband transmissions.[4] Unfortunately, Manchester encoding is only 50 percent efficient, and its use in FDDI's 100M-bps data rate would have required 200 megabauds on the medium, with the LEDs and pin receivers operating at 200 MHz.

FDDI uses a more efficient encoding scheme, called 4B/5B, to keep the baud rate down.[5,6] In 4B/5B, the encoding is performed on four bits at a time to create a five-cell "symbol" on the medium. It is 80 percent efficient, and at 100M bps the baud rate becomes 125 megabauds. The advantage is that inexpensive LEDs and pin diode receivers, which only have to operate at 125 MHz, can be used.

Token-passing ring

FDDI uses a token-passing ring consisting of stations serially connected by a transmission medium to form a closed loop. The packets are transmitted sequentially from one station to the next, where they are retimed and regenerated before they are passed on to the following (or downstream) station. The idle stations can either be bypassed or can function as active repeaters. The addressed station copies the packet as it passes by. Finally, the station that transmitted the packet strips the packet off the ring.

A station gains the right to transmit when it has the token, which is a special packet that circulates on the ring behind the last transmitted packet. A station wanting to transmit captures the token, puts its packet(s) on the ring, and then issues a new token, which the next station can capture for its transmission.

Access scheme. Using an access scheme called timed-token access, FDDI allows each node a fair share of access to the network and at the same time maintains an upper bound on the token-rotation time. The token-rotation time determines how often the nodes get an opportunity to transmit.

To satisfy the requirements of various applications, a node needs to know two things. One is the maximum latency before the token returns again to the given node. The other is the amount of traffic that the node can send once the token returns. The FDDI protocol allows the node to determine both these parameters through negotiation.

The total bandwidth available on the ring can be dynamically partitioned off as either synchronous or asynchronous bandwidth.[5,6] Here, synchronous refers to bandwidth that is guaranteed to a node for its use, every time it gets the token. The leftover bandwidth is the asynchronous bandwidth and is shared by all the nodes.

Two kinds of packets can be transmitted on FDDI. Synchronous packets can be sent by the nodes that have negotiated the synchronous bandwidth, every time the token is received. However, asynchronous packets can be sent only if the token is received early (that is, within the agreed-upon upper bound). The asynchronous packets can be one of eight priorities. Which priority to send is determined by a threshold time value.

FDDI also supports a special token class called restricted tokens. This class is provided mainly for back-end networks having devices that need to hog the network for data streaming.

Benefits to various applications

As shown in Figure 1, FDDI can be used as back-end, backbone, and front-end networks. Different features of FDDI make it attractive for each application.

Back-end network. For a back-end network in a computer room, reliability and performance are the important considerations. In this environment, it is desirable for two stations to maintain unimpaired operation, even if up to six intervening stations are powered down (causing their optical bypass relays to be in the active connection path between the communication stations). FDDI is projected to meet this constraint when the total fiber length between communication stations is less than 300 meters.

The dual rings, by reconfiguring, provide additional fault tolerance. In a back-end network most nodes will be Class A nodes talking to both rings. If the 100M-bps data rate is not sufficient in this application, an additional 100M bps is also available on the secondary ring as long as the ring is not reconfigured.

FDDI also supports a class of service called restricted token. This sets up a master/slave connection between the host computer and a peripheral controller, allowing the two to hog the network and stream data between them. Although the restricted token violates the deterministic access of a token ring, it is acceptable in a back-end environment.

Backbone network. A backbone network stretching over a campus may span several kilometers. The backbone can be viewed as a collection of trunk lines connecting several computer rooms or individual networks in the office environment or distributed PBXs. A high bandwidth on the backbone ensures that it is not a bottleneck during internetworking. FDDI is designed to allow links at least two kilometers in length between adjacent nodes with no optical

bypasses on either end. The total allowable fiber-path length to satisfy the default-timer values and ring-latency constraints is 200 kilometers. Since duplex cable is used, the actual length of cable in the entire ring is 100 kilometers. The 100M-bps data rate provides ample bandwidth for this application.

Another concern during internetworking is station addressing. Since FDDI permits both 16-bit and 48-bit addresses with physical and logical addressing support for both as well as broadcast capability, the addressing translation across gateways and bridges is simplified.

For supporting distributed PBXs, a circuit-switched or time-slotted access method is desirable. The ANSI X3T9.5 committee is starting preliminary work on the FDDI-2 specification, which will implement a combination of circuit- and packet-switched traffic. This will make it easy to provide a gateway to the telephone network or to the ISDN.

Front-end network. In the office-floor situation FDDI is suitable as a front-end network. The cost of connection can be minimized by having a preponderance of Class B stations. These Class B stations can be attached to wiring concentrators, which in turn will provide Class A connections to the backbone FDDI network.

Because wiring concentrators generally are powered, the need for optical bypassing is minimized. Class B connections lower the cost of the connection since the duplicate physical layer logic is not needed. At the same time, they still form a part of the main ring and can transfer data at the 100M-bps data rate.

FDDI provides for a separation of up to 500 meters between a Class B station and a wiring concentrator. When any of the Class B stations is powered down or its link is broken, the wiring concentrator can bypass it electrically very easily. The wiring concentrator also provides a convenient maintenance point.

The FDDI committee has made reasonable trade-offs to ensure that the technology needed for implementing FDDI is presently available and that the standard will not be obsolete in the next several years. The standard's authors have kept in mind upgradability to higher speeds and additional services that will be necessary in the future.

The emergence of FDDI as a popular standard is spurring integrated circuits development from semiconductor vendors. The use of dedicated VLSI to support FDDI will lower the cost of an FDDI interface dramatically.

References

1. Sunil Joshi, "Session Overview: A look at Standards Committees," *Proc. Wescon 84,* Oct.-Nov. 1984, pp. 1-5.
2. David Berry, "Standardizing Upper-Level Network Protocols," *Computer Design,* Feb. 1984, pp. 174-190.
3. *Draft Proposed American National Standard FDDI Physical Layer Protocol (PHY),* ANSI X3T9/85-39, X3T9.5/83-15; *FDDI Media Access Control (MAC),* ANSI X3T9/84-100, X3T9.5/83-16; *FDDI Physical Medium Dependent Layer (PMD),* ANSI X3T9/85, X3T9.5/84-48; and *FDDI Station Management (SMT),* ANSI X3T9/85-, X3T9.5/84-49; American National Standards Institute, New York, 1979.
4. V. Iyer and S. Joshi, "Hardware Considerations in Local Area Networks," *Proc. Midcon 82,* Nov. 1982.
5. Sunil Joshi, "Making the LAN Connection with a Fiber Optic Standard," *Computer Design,* Sept. 1985, pp. 64-69.
6. S. Joshi and V. Iyer, "New Standards for Local Networks Push Upper Limits for Lightware Data," *Data Communications,* July 1984, pp. 127-138.

Sunil P. Joshi is the section manager for local area networks at Advanced Micro Devices. His duties include the planning and specification of high-speed LAN products. During the past six years at AMD he has participated in the design and development of several Am2900-family microprogrammable products, the Ethernet chip set, and the Supernet chip set designed to implement FDDI.

Joshi holds a BSEE from the Indian Institute of Technology, Bombay, and an MSEE from Rensselaer Polytechnic Institute, Troy, New York.

Questions about this article can be directed to Joshi at Advanced Micro Devices, Inc., 901 Thompson Place, PO Box 453, Sunnyvale, CA 94086.

This device has been designed to provide IEEE 802.4 media access control in factory networks employing the GM MAP protocol.

The MC68824 Token Bus Controller

VLSI for the Factory LAN

Rhonda Alexis Dirvin and Arthur R. Miller
Motorola, Inc.

The MC68824 token bus controller implements the media access control, or MAC, function for the IEEE 802.4 token bus. The MAC function of a protocol encompasses the most time-critical facilities required in a communications link and represents the major portion of the data-link-control layer of the seven-layer Open Systems Interconnection, or OSI, communications model of the International Standards Organization.

VLSI implementation of a controller for the IEEE 802.4 token bus is particularly timely since token bus has been specified as the MAC layer for the Manufacturing Automation Protocol, or MAP. (See "Token bus is best for factory LANs," next page.) First demonstrated last November at Autofact '85 in Detroit, MAP has been championed by General Motors and other companies that are becoming dependent on industrial automation. MAP represents a user-defined subset of the OSI model that is specified as a local-area network for factory automation (Figure 1).

Here, we will show how a VLSI MAC controller reflects both the characteristics of the protocol implemented and the method of system segmentation chosen

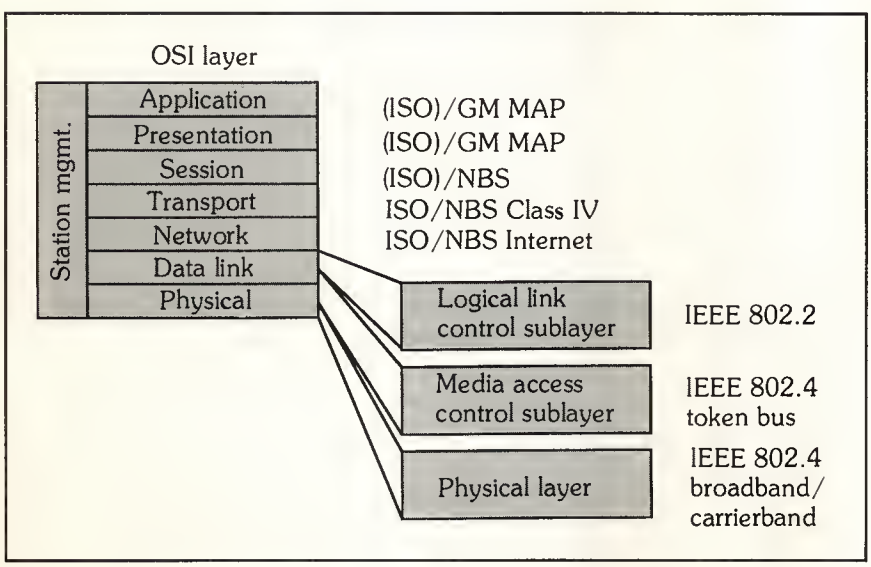


Figure 1. The ISO Open Systems Interconnection Model—the various layers and sublayers of the model have been standardized in GM MAP, IEEE 802.4, and other specifications. The MC68824 implements the IEEE 802.4 media access control sublayer, which has also been specified as the media access control sublayer for MAP.

0272-1732/86/0600-0015$01.00 © 1986 IEEE

Token bus is best for factory LANs

Several features can be used to separate local-area network implementations into the market areas each can best serve. Two characteristics make a token bus a better choice for a factory LAN than other standards—topology and method of media access. The choice of the medium itself is significant as well but is less a factor than topology or access method since the primary media types—coaxial cable, twisted-pair cable, and optical-fiber cable—are, or will be, available for the various LANs.

Topology. The topology of a network is the physical and electrical implementation of that network. The three implementations typically considered are the ring, the star, and the bus (Figure A).

In a ring, information transmitted by one of the stations is routed through all the other stations. The information carries an address identifying it to its intended destination. This means that most of the stations will simply receive and retransmit the information since it is not intended for them. A major problem of a ring in a factory environment is that a failure in any one station can bring the entire network down. Adding or removing a station also brings the network down. Designs that attempt to overcome this problem through the use of additional hardware such as wiring closets and star-wired bus connections can drastically increase the cost of a factory network. Another problem of a ring is that it does not fit the typical linear structure of an assembly line.

A star topology also has the problem of not fitting a typical factory layout. A star configuration requires a large amount of wiring, since each station must be individually connected to a central node. Once in place, this wiring makes it difficult to reconfigure assembly lines for new products or more efficient operation. (Note that the central node in a star topology can function as an active router or merely as a point of passive interconnection.)

A bus topology fits the typical factory layout. It allows taps wherever they are needed to connect manufacturing cells. It allows stations to be added or deleted with relatively little impact on the network. It also allows control of the network from any point on the LAN, providing a true distributed control capability. The bus topology can allow an ordered rotation of the ability to transmit data on the network; that is, it can permit the formation of

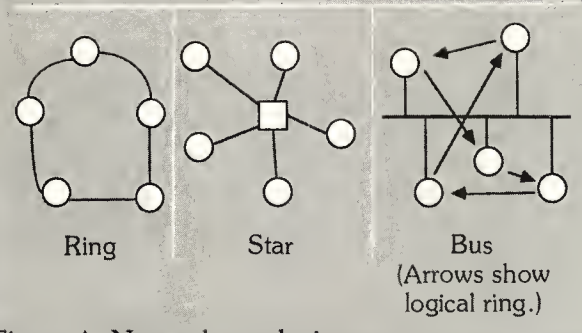


Figure A. Network topologies.

by the designers to fulfill the requirements of the LAN specification.

The device described, the MC68824 token bus controller, has been implemented in two-micrometer HCMOS. It interfaces both to a software logical link control, or LLC, sublayer and to a physical layer. Because MAP requires more than one media type and data rate for different applications within the factory—all of which require MAC functionality—the IEEE 802.4 committee recommends a standard MAC-to-physical serial interface, which the MC68824 implements. By meeting this interface specification, the MC68824 can act as a single interface point to the physical media, both for normal data transmission and reception and for the passing of station management information between the modem and the host processor. The MC68824 therefore can support all IEEE 802.4 data transmission rates—one, five, and ten megabits per second—and both broadband and carrier-band modulation.

Implementation and internal architecture

To enable system designers to provide a configurable MAC function, to support a high-performance 10-megabit-per-second serial interface, and to be able to respond quickly to changes in a protocol that was still being defined while the VLSI was in design, the MC68824's designers chose a microcoded architecture with the internal block structure shown in Figure 2. This resulted in an HCMOS device of 82,000 actual transistors, with 2.5K 22-bit control words in the microcontroller.

By using a microcoded machine to control the other components on the chip, the MC68824's designers were able to change the major functionality of the part almost up to the time the device was committed to silicon. The simple microcoded structure also made the part easier to test. Special control structures on the chip, such as a DMA control block, were used by the designers, as

a *logical* ring (see Figure A again). Formation of a logical ring is accomplished through the choice of media access method, as occurs in the token bus scheme.

Media access control. Two methods of media access control are currently allowed under IEEE standards—carrier-sense multiple access with collision detection, or CSMA/CD, and token passing.

CSMA/CD relies on each station to listen to the LAN for the presence of a carrier indicating that another station is transmitting. A station may begin its own transmission when it senses that the network is free, although it must still listen for other stations that may begin to transmit at essentially the same time. If this occurs, it is a collision and all stations involved immediately cease transmission. Each station then waits a randomly generated length of time for the network to become free again. The random-length time period helps ensure that the collision will not occur again.

CSMA/CD is the type of media access specified by the IEEE 802.3 standard. It has been found to perform quite well in typical office environments, where communications traffic occurs in bursts and where there is little need to ensure that data will be transmitted within a certain period of time. However, the performance of CSMA/CD networks rapidly degrades as communications traffic reaches a network-dependent threshold. Thus, CSMA/CD is inadequate for a factory control environment, where the traffic is relatively steady and can be consistently heavy and where the delay of control information can result in physical damage to systems.

In token-passing media access, each station in a physical or logical ring is given a turn to transmit data. A station's turn comes up when it receives a token from the previously transmitting station. When it finishes its transmission, the station passes the token to the next station in the ring. This arrangement ensures that each station will be able to access the network within a given amount of time. When a maximum period of time that a station can hold the token can also be defined, a deterministic network can be designed. In such a network, the maximum amount of time between successive receipts of the token by any one station can be well defined. If this maximum time is shorter than the shortest critical time period required in a distributed control application, then the network can be said to provide real-time capability.

The above considerations led General Motors, in the MAP specification, to the conclusion that a local-area network with a bus topology and a token-passing method of media access best fits the distributed control environment of the factory. A token bus scheme not only meets needs associated with the physical layout of a typical factory but also supports deterministic implementations that can meet real-time control requirements.

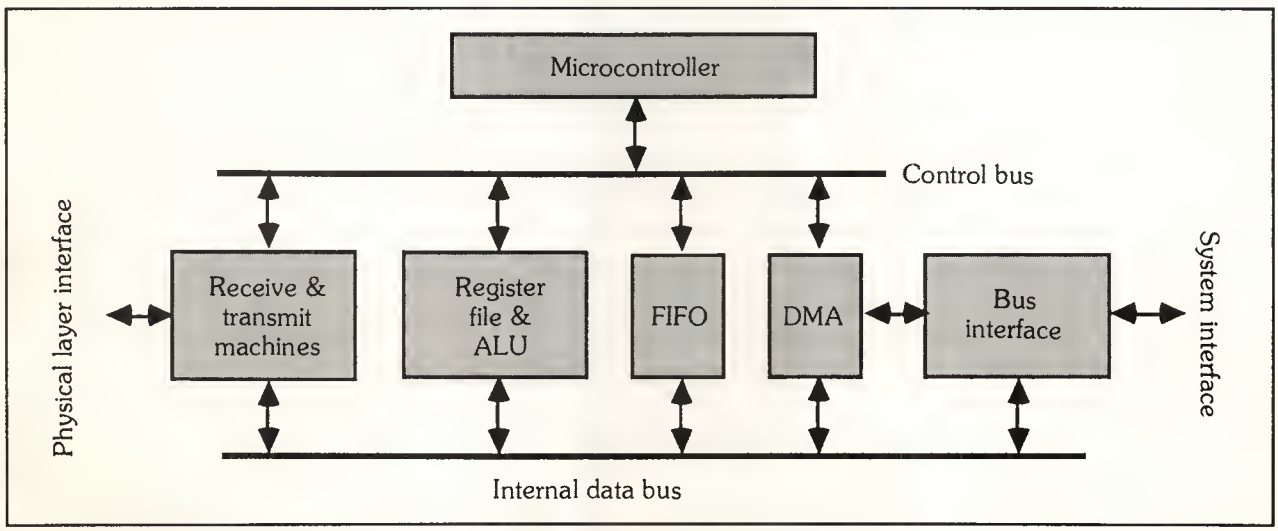


Figure 2. The internal architecture of the MC68824 token bus controller.

necessary, to enhance performance. This DMA block, for example, operates with the bus interface to implement a four-channel direct memory access function.

To support high serial data rates and heavy DMA operation, the MC68824's designers implemented a separate DMA state machine that can be programmed by the microcode machine and left alone to do data transfer functions while the microcode sequencer is busy performing other parts of the protocol. Data transfers are usually between system memory and the FIFO or register files.

To ensure that the chip would operate reliably in a system at a 10-megabit-per-second data rate without overruns on reception or underruns on transmission, its designers incorporated a 40-byte FIFO into it. This FIFO size, in conjunction with microcode functionality and DMA flexibility, typically provides a minimum of 25.6 microseconds (32 bytes × 8 bits/byte × 0.1 microsecond per bit) of system latency before a fault occurs on initial access.

The main DMA features supporting the system designer are

- a choice between unlimited data transfer cycles for the FIFO to/from memory or a limited burst mode whereby the token bus controller relinquishes the bus and rearbitrates after eight memory cycles (two bytes or two words transferred in 8-bit mode or 16-bit mode, respectively),
- ability to operate as a 16-bit data path device or an 8-bit data path device, and
- a delay in starting transmission of a frame or in requesting the bus upon reception of a frame, until the first six bytes of the message are in the FIFO.

MC68824 DMA supports 16-bit data transfers in either IBM/Motorola byte order or DEC/Intel byte order.

The bus interface chosen for the MC68824 has MC68000-type signal definitions but, since a choice of data byte order is allowed, it can also be configured, with a minimum of glue logic, for operation with 8086-type devices. The bus interface provides a function code capability to enhance system software protection schemes. The main bus itself provides a full 32-bit address in addition to the 16-bit data path. This extensive addressing capability works well either with high-performance nodes operating with dedicated processors or with remote nodes in a low-performance system. Neither the data nor the address lines are multiplexed. This saves glue logic for the system implementer and makes continuous four-cycle DMA possible.

When combined with devices designed for system bus clock frequencies of up to 12.5 MHz, the MC68824 can maintain transfer rates of up to 50 megabits per second between the LAN controller and the system memory during message reception and transmission. This means that the system designer can choose options that will leave at least 70 percent of the system bus's bandwidth available to the local processor for implementing upper layers of the protocol in software. This makes it possible to provide the functions of all seven layers of the OSI model on a single, medium-sized, printed-circuit board.

The MC68824 incorporates receive and transmit machines that implement a cyclic redundancy check, or CRC, function and effect serial-to-parallel and parallel-to-serial conversion. These machines can also be used in conjunction with the microcode sequencer and special commands to act as a pass-through between the system and the physical-layer modem for control functions.

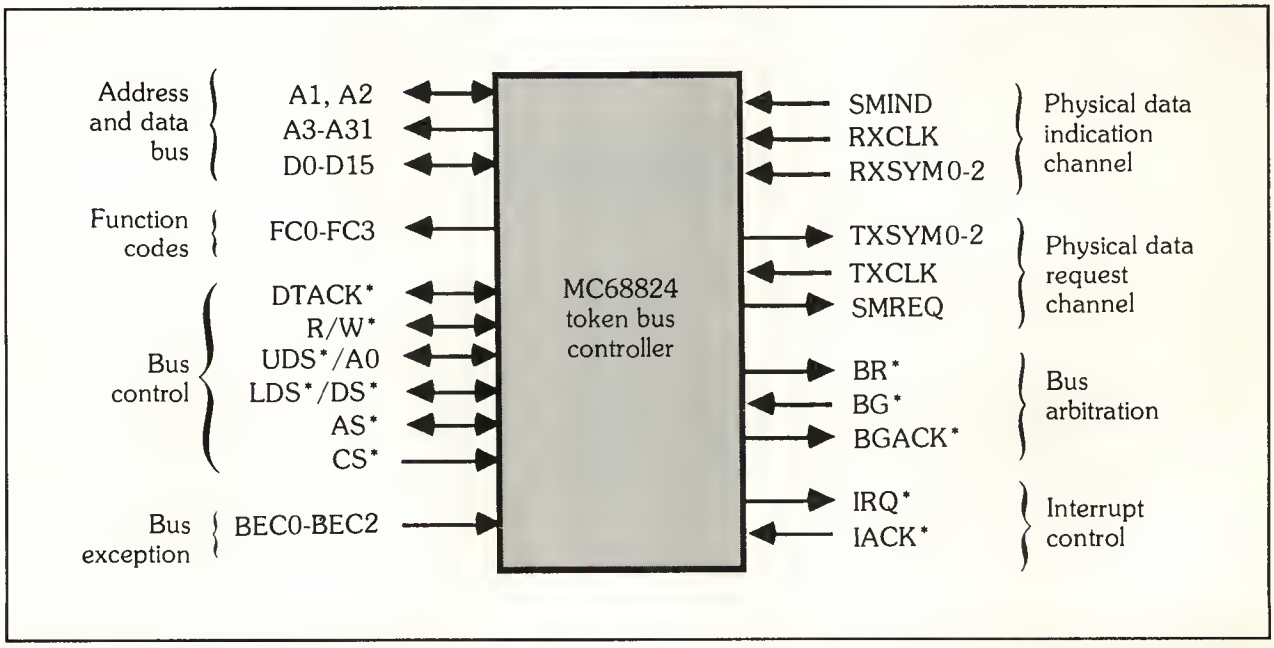


Figure 3. Pinout of the MC68824 token bus controller, showing the bus functions associated with the pins.

The MC68824 ALU performs address generation for system functions, address comparison for network addresses, and statistical collection for the network. Most of the addresses and statistics are maintained off chip in a special 256-byte block of memory known as the token bus controller private area.

There are three registers on the MC68824 available to the user. These registers—the command register (CR), the interrupt vector register (IV), and the data register (DR)—are used to load addresses, commands, and other setup parameters into the device prior to initialization. Having registers for these functions on chip gives board-level designers flexibility in allocating memory space to peripheral components. The command register also functions in the read mode as a semaphore register that indicates when the MC68824 is ready to accept a new command. If a command requires more than four clock cycles to complete, this register should be checked before another command is issued. The user selects the registers by employing the lower address lines (A1 and A2) and the data strobes (UDS* and LDS*) as addressing mechanisms when the chip select (CS*) is activated.

The pinout of the MC68824 (Figure 3) shows the major bus functions of the device. We should note for those unfamiliar with M68000 processors and peripherals that function codes are used in the MC68824 as extensions to the 32-bit addressing mechanism. They are useful in separating processor and peripheral address spaces and can be used to designate a different memory space for each of the MC68824's memory structures (to be detailed in the section on shared memory, to follow). If there is no need for function codes in the system being designed, they can be ignored.

Commands

The MC68824's commands are tailored to the token bus communication protocol. They are used for initialization, for changing parameters and modes during typical device operation, and for management functions. Besides providing these functions, the MC68824 acts as a high-performance protocol controller and, once initialized, frees the host from having to intervene in the normal transmission and reception of frames.

The MC68824's command set includes commands to

- initialize the device,
- set its mode of operation,
- transmit data frames,
- set a parameter and read its value,
- test the device,
- notify the device, and
- control the modem.

Initialization commands are used to load parameters from the initialization table in shared memory both into the MC68824 and into the MC68824's private area in shared memory. Before an initialization command is issued, the pointer to the initialization table is loaded into the DR register so that when the initialization command is issued the MC68824 knows where to obtain initialization data. These commands also can change the state of the MC68824 from off-line to idle, enabling it to actively participate in the network. A reset command is also included in the initialization command category.

"Set operation mode" commands are used to choose different options and modes in the MC68824. These options and modes enable the device to be easily configured for any one of many diffferent token bus networks.

Some of the options and modes are meant to be set for the entire length of the station's life in a specific application. Examples of such options and modes are a host processor data bus width of 8 or 16 bits, a network address length of 16 or 48 bits, a "copy all control and data frames" mode (for use by a station that is a promiscuous listener, such as a station that is used in a network monitor), and network bridging and routing modes (see "Bridging and routing," next page). Other options and modes meant to be set for the life of a station include a mode that enables gathering of network performance statistics, a DMA burst mode, a swap mode, which causes word-length data in the data buffers to be organized into bytes, and a prescaler option for most of the timers.

Other options can change during network operation. Examples include copying or not copying a received data frame's CRC to the memory as data, enabling or disabling a transmitter's CRC, and changing the "in ring desired" bit, which, if set, makes the station a part of a logical ring. (If "in ring desired" is not set, the station becomes part of the logical ring only if it has something to transmit. If it does, it transmits it and then exits.) Other examples of options that can change during network operation include a predefined response mode and a mode that allows frames with undefined frame control to be recognized.

The MC68824's options and modes, both those that are set for the life of a station and those that can change during network operation, allow the station to be set up as, among other things, a gateway, a network manager, a bridge, an active node, or a passive node.

The "transmit data frames" commands are used to stop, start, and restart the transmission of data. These commands do not have to be issued often. The start command, for example, is issued only when a frame is added to an empty transmission queue. The stop command is issued only when the host wants to halt the transmission of frames. The restart command is used to continue transmission after a stop command.

The "set parameter value" and "read parameter value" commands are used to set and read the values of parameters that are not directly accessible to the host, i.e., that are located either in the private area or inside the MC68824. These parameters are set at initialization and typically do not need to be changed. This keeps the interface to the host simple and the number of directly accessible registers to a minimum. These parameters are set or

Bridging and routing

Network implementation often requires the joining of several networks having different data rates, media, or broadband channels. Sometimes, however, it requires the interconnection of identical LANs; interconnection of such networks affects critical network parameters less than interconnection of disparate networks. Token bus networks are especially likely to be interconnected, since token bus network standards allow both broadband and carrierband (frequency-modulated baseband) implementations. For example, a carrierband token bus may be used within manufacturing cells because it is cheaper, whereas a broadband token bus may be used to implement the factory backbone because it provides higher data rates, multiple channels, and better noise immunity. And multiple carrierband token bus networks may be connected so as to keep their respective token rotation times short.

Systems that interconnect networks are called *bridges*, and the MC68824 token bus controller has been provided with features that enable it to aid in implementing the bridge function. The MC68824 supports two types of bridges—the hierarchical bridge and the flat bridge.

The specific MC68824 mechanism that aids in flat bridging is referred to as IBM-defined source routing. Source routing determines the best routing between a transmitting station and its target station. Once this routing is known, it is used in all subsequent communications between the stations. Source routing is specified under IEEE 802.1.

Hierarchical bridging. An example of a hierarchical bridge is given in Figure B. A hierarchical bridge includes both an upper and a lower bridge function. An upper bridge accepts and passes on all frames destined for segments below it in the hierarchy, whereas a lower bridge accepts and forwards all frames destined for segments above it in the hierarchy. When the MC68824 is programmed to the proper operation mode, it accepts all frames for bridging in which

$$IA \cdot DA = IA \cdot TS, \text{ where}$$

IA :: = the individual address mask of the bridge,
DA :: = the destination address of the received frame, and
TS :: = the receiving station's address.

This is illustrated by the figure and by an example. (Note that the MC68824 can be programmed to recognize either 16-bit or 48-bit addresses. In the example we assume that only the lower 12 bits will have different values on the backbone bus. The upper bits in the IA mask will be set to 1 and will match because of the way the LAN backbone is defined; that is, frames will be passed to this backbone by another bridge only if all the bits above the lower 12 bits are identical.)

Assume that bridge B's IA mask is set to F00 (i.e., FF00 or FFFF FFFF FF00) and a frame ar-

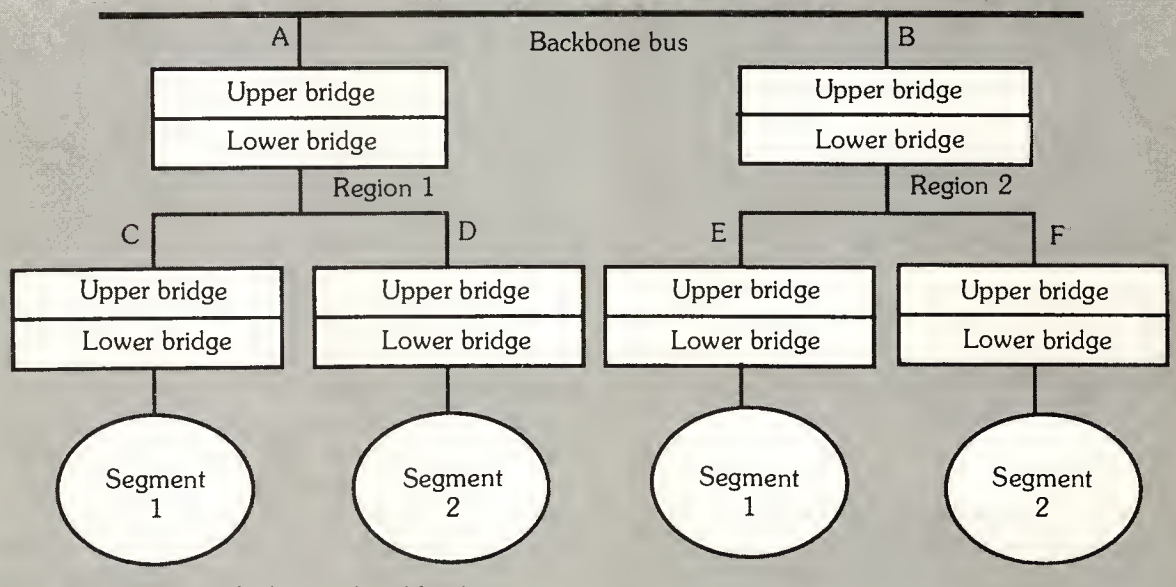


Figure B. Structure of a hierarchical bridge.

rives addressed to station 2, segment 1, station 4 with an address of 0010 0001 0100:

IA	1111 0000 0000	IA	1111 0000 0000
DA	0010 0001 0100	TS	0010 xxxx xxxx
IA·DA	0010 0000 0000	IA·TS	0010 0000 0000

Since the equations generate an equality, the frame is accepted by bridge B for transmission to region 2.

Also assume that the lower bridge of bridge F has also been implemented by an MC68824 but that the "lower bridge mode" bit has been set. Furthermore, assume that a frame has been generated on segment 2 of region 2 and that it is addressed to station 215. The example as generated by the lower bridge of bridge F will be

IA	1111 1111 0000	IA	1111 1111 0000
DA	0010 0001 0101	TS	0010 0010 xxxx
IA·DA	0010 0001 0000	IA·TS	0010 0010 0000

In this case, since the equations do not generate an equality and the "lower bridge mode" bit is set, the frame is accepted for transmission to region 2. Note that the result would have been the same had the frame been addressed for a destination in any segment other than segment 2 in region 2.

Note that for hierarchical bridging structures to work properly, some thought must be given in advance to address assignment for the networks.

Flat bridging. Flat bridges interconnect segments in any fashion. For efficient flat bridging, the best routing to destination stations is first determined. This routing information is then retained in a table of translations by the transmitting station, for subsequent use. In flat bridging, each MC68824 in the bridge has the following parameters:

SID :: = Source ID. This is the segment number (SN) of the segment to which the MC68824 is directly connected.

TID :: = Target ID. This is the segment number (SN) of the segment connected to the other MC68824 in the bridge.

SR-MASK :: = This is a two-octet parameter defining (by bits set to 1) which bits in the SNs comprise the segment number.

Source routing must be selected in mode 2 operation. When this occurs, the setting of the first bit of the source address (SA) indicates the presence of a routing information (RI) field preceding the data field of the frame. This first bit of the SA, when used in this fashion, is referred to as the routing information indicator, or RII. (Note that IEEE 802.4 reserves this bit for future use.) The RI field then follows and contains from 2 to 32 octets. The first two octets act as a control field that indicates

- whether the frame is a resolve frame destined for all LANs ("broadcast" bit set to 1),
- the total length of the RI field, including routing control octets (maximum 32 octets), and
- the direction bit.

A direction bit set to 0 indicates travel in the same order as that of the segment numbers received, whereas a direction bit set to 1 indicates travel in an order the reverse of that of the segment numbers received.

To establish a preferred routing to a target station on another LAN, the transmitting station sends a resolve frame with the RII bit set to 1, the broadcast bit set to 1, and the direction bit set to 0. The source address is the SA of the transmitting station, and the destination address is the DA of the target station. This frame is then rebroadcast by every bridge on the LAN. At each bridge, the RI field is scanned for the SN of the TID. If that SN is there, the bridge will not forward the frame across itself, since that frame has already been in that segment. If the TID is not in the RI field, the bridge adds the TID to the RI field and increments the subfield of the routing control octets by two before forwarding the frame to the TID.

As differently routed frames arrive at the target station, the target station will select 1 and respond by swapping the DA and SA of the received frame, changing the direction bit to 1, and changing the broadcast bit to 0. The received frame is thus routed back to the original transmitting station in the reverse of the original route. The original transmitting station then adds to its table of translations the routing information from this response frame. This information is the routing information used in subsequent transmissions to the target station.

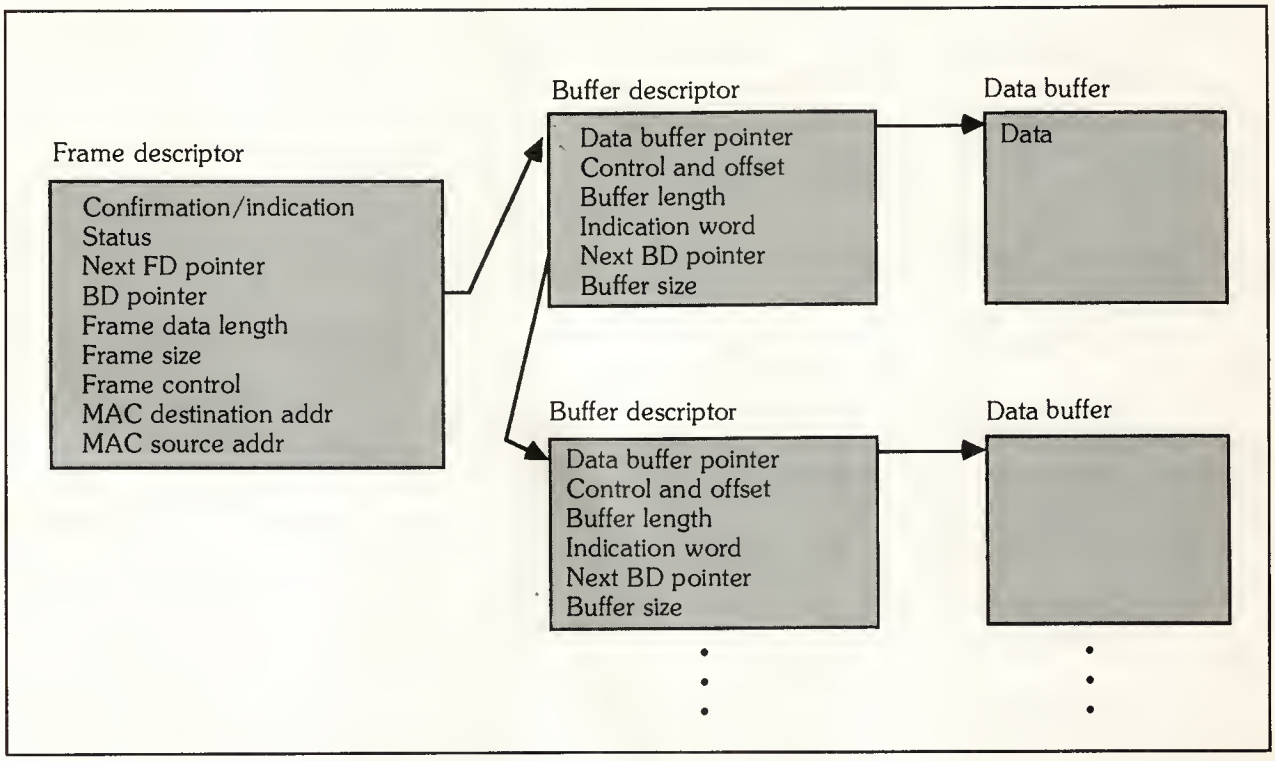


Figure 4. Structure of the linked data buffer.

read by placing the desired parameter's code in the initialization table and issuing the appropriate command.

The test commands initiate one of four tests. These tests allow the user to test the MC68824 in a system to make sure that its major parts and the modem are working.

The "notify" commands are used to notify the MC68824 either that an interrupt has been handled and the interrupt request line must be cleared or that a response to a request-with-response frame is ready (see "Extensions," below).

The modem control commands are used to put the modem into or take it out of station management mode. In station management mode, control information is passed between the physical layer and the host through the MC68824.

Shared memory

To support the high speed of a token bus network having potentially large data transfers to and from a host, the MC68824 utilizes a shared memory. Structures within this shared memory include a linked data buffer structure, the initialization table, and the MC68824 private area, which is used to store internal variables and parameters.

The linked data buffer structure. A fully linked data buffer structure (Figure 4) provides data transfer service to the logical-link-control sublayer while allowing efficient use of system memory. The data buffer structure is made up of a frame descriptor (FD), buffer descriptors (BD), and data buffers. The frame descriptor contains control information and frame attributes. This frame descriptor points to a buffer descriptor, which in turn points to a data buffer in which the data are kept. There is a buffer descriptor for each data buffer. If more than one data buffer is needed because the frame is too long to fit into a single data buffer, buffer descriptors are linked.

A useful feature of the MC68824's data buffer structure is an offset in the data buffer. The buffer descriptor maintains an offset byte count and tells how far from the beginning of the buffer the actual data begin. This offset feature is extremely useful for building data frames as they are passed down through the layers of the OSI structure. Addresses and control information can be appended to and removed from the front of each frame as it progresses through each software layer, making it unnecessary to actually move the data.

Data buffers do not need to be contiguous in memory, and the data buffer size can be programmed by the user. These attributes make flexible memory management possible. A programmable data buffer size is another means of tailoring a network to a specific application. The expected network frame length can determine the most efficient data buffer size.

Tables. The shared memory area contains two tables—the initialization table and the private area. The initialization table contains the initial values of parameters for an IEEE 802.4 token bus such as the slot time and the ring maintenance initial value. It also contains pointers to the beginning of each queue and a command parameter area that is used to set and read parameters within the MC68824 that are not accessible through directly addressable registers. The initialization table is also where statistics are kept if the statistics option has been chosen.

The private area is used as an external scratch pad for the MC68824. Note that to keep the die size manageable and yet offer a wide range of options, the MC68824's designers arranged for some parameters to be maintained off chip until they are needed or need to be updated, in which case they are automatically fetched by microcoded functions through the DMA block. The overhead involved in keeping such parameters and table structures off chip is low and typically represents less than 10 percent of the system bandwidth available after pure data transfer requirements have been met. The private area is not meant to be read or changed directly by the host, since such a partial update can violate the workings of an IEEE 802.4 node. Many parameters in the private area are initialized through the initialization table and can be changed through MC68824 commands.

Extensions

The MC68824 implements several features beyond the basic IEEE 802.4 requirements:

- four classes of message priority and a request-with-response mechanism (these are real-time, IEEE-specified extensions),
- monitoring capabilities (these allow the user to monitor potential error conditions in the network),
- diagnostic aids (these are accessed through the test commands and help isolate problems in the modem, transmitter, receiver, or MC68824 host interface), and
- addressing modes (these implement a group and individual addressing capability).

Priorities. The IEEE 802.4 protocol contains an option for four classes of message priority. The MC68824 supports all four of these classes, each of which can be individually enabled or disabled. A priority system can be extremely important in a factory LAN, since in such a LAN both routine message traffic—such as batch information messages—and high-priority message traffic—such as catastrophic failure messages—may be generated. The MC68824 implements the priority mechanism by linking descriptors of frames within the same priority queue and providing token rotation timer functions for each priority class. There are four separate queues for received frames and four separate queues for frames to be transmitted.

Request-with-response mechanism. The MC68824 implements the IEEE-approved request-with-response, or RWR, mechanism to support real-time networks. Such networks include ones for process control and ones which employ MAP's enhanced performance architecture, in which network services can be sacrificed for speed of message transfer. Real-time networks require the MC68824

- to be able to guarantee that each station on the network will be given an opportunity to transmit within a relatively short period of time (tens of milliseconds),
- to be able to maintain a list of active transmitting stations on the network, and
- to provide a message service using RWR that allows a destination station to respond immediately to a request frame sent by a station with the token.

An RWR frame effectively "sublets" the token to a destination station so it can respond to a request within one return frame. The MC68824 supports the RWR frame type by allowing the user to employ a predefined response mode in which a pointer to a prepared response can be maintained and in which the MC68824 can send that response without host intervention. Use of this mode reduces the time it takes a requested station to respond. In non-predefined response mode, an interrupt is generated when the RWR frame is received and the LLC must prepare the appropriate response. When the response is ready, the host issues a "response ready" command to the MC68824, which then sends the response to the requesting station.

Monitoring capabilities. The MC68824 monitors itself, the modem, and the network for error conditions by collecting statistics. By using these statistics along with the "copy all frames" mode, a station can determine characteristics of the network and message traffic.

Each statistic has a threshold value associated with it. This mechanism allows the host to set a value that a statistic must reach before the station collecting the statistic can generate an interrupt. With this mechanism, the host can either monitor problem areas in the node or map the MAC counter (16 bits) to a larger counter (32 bits) in upper-layer software. If the network designer does not want to use the threshold mechanism for a statistic, he has the host set the appropriate threshold counter to zero.

The statistics the MC68824 tracks, and some examples of the way the host uses the statistics, include the following:

- The total number of tokens passed. The host uses this number to estimate the number of stations in the logical ring.
- The total number of tokens passed on the network by the station-enabling software. The host uses this number to calculate the average token rotation time.
- The number of times the MC68824 fails to pass the token and does not succeed in finding a new successor sta-

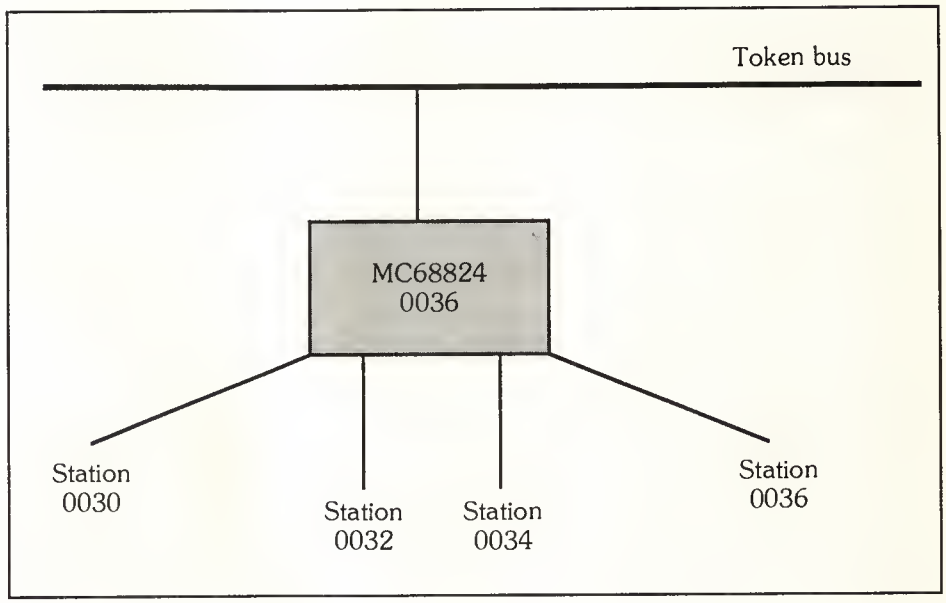


Figure 5. The four tests the user can employ to check systems using the MC68824.

tion. This may indicate a faulty transmitter in the station.

• The number of times a station has had to look for a new next station to which to pass the token and the number of token passes that have failed. The host uses these statistics to determine the stability of the logical ring.

• The number of frames greater than 8K bytes (the IEEE 802.4-specified maximum frame length). The MC68824 can transmit frames of up to 64K bytes, a capability useful in networks that must support the downloading of software.

• The number of frames that were not received because there were not enough frame descriptors or buffer descriptors. This indicates station management problems.

• The number of times the FIFO has overrun during receive operations. This indicates a need to increase the priority of the MC68824 in the host system.

Modem error statistics include the following:

• The number of frames received with no start delimiter.

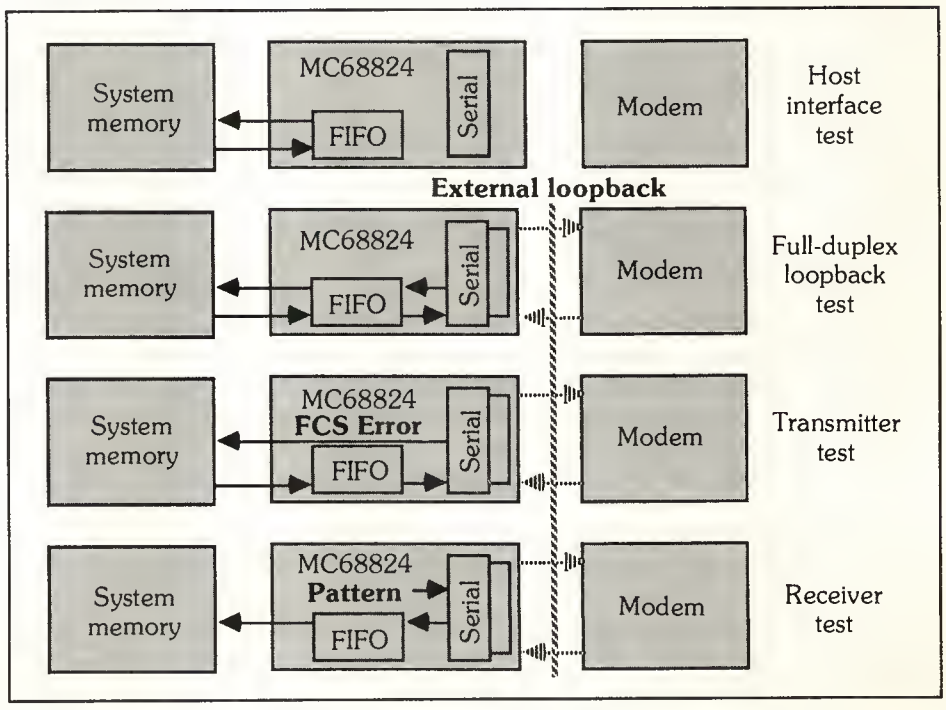


Figure 6. Token bus multidrop node. Such a node is implemented through individual address masking.

• The number of frames received with a CRC error bit. The error bit is set by the head-end remodulator in a broadband network if the head end finds an error in the CRC.
• The number of error-bit errors.
• The number of frame fragments.

Diagnostic aids. Four tests for systems using the MC68824 are available to the user: a host interface test, a full-duplex loopback test, a transmitter test, and a receiver test (Figure 5). These tests are unique in that they can test all the internal workings of the MC68824 even though the device is a half-duplex implementation.

The host interface test is used to test the path from the host through shared memory to the MC68824 and back.

The full-duplex loopback test checks all the things the host interface test does, as well as the FIFO and the entire serial portion of the MC68824. It can also check the modem. This test is performed as follows: The first half of a 72-byte data buffer is filled with test data and the appropriate test command is issued. The MC68824 then returns the test data to the second half of the 72-byte buffer after transmitting from half of the FIFO upon transmit, looping through the serial machine, and filling up the other half during receive. If the external loopback mode is selected, the data also go through the modem.

In the transmitter test, test data are placed in data buffers that are looped together or placed in straight lists. In this test, data are transmitted out normally. The only receiver check is for an error in the transmitted CRC. The transmit CRC can be turned off. In this case, the user must calculate his own CRC. This enables the user to check the CRC generator for proper operation.

The receiver test checks the receiver by generating a pattern that causes the receiver to work normally. This pattern is compared to the pattern received by the station host.

Addressing modes. The MC68824 can recognize group-addressed frames and frames for stations other than itself. Group addressing allows a large number of logically related stations to receive the same message. Messages can be sent to just one group, but a station can belong to more than one group. The MC68824 recognizes a group address when the least significant bit of the destination address (the I/G bit) is equal to 1. The MC68824 also provides a multidrop capability by means of which several users can share the cost of a high-performance node. This is implemented through an individual address mask option and is illustrated in Figure 6. (See "Bridging and routing," page 20, for more details.)

We have described a VLSI token bus controller for an industrial local-area network. The device supports the IEEE 802.4 standard for media access control, as it is specified for the Manufacturing Automation Protocol, and it offers important extensions to that standard that MAP may eventually use. It will provide the functions and performance needed by nodes in a factory LAN, doing so with the lower parts count and lower cost implicit in VLSI.

Bibliography

IEEE Standard 802.4-1985, Local Area Network (Token-passing Bus Access Method and Physical Layer Specifications), Institute of Electrical and Electronics Engineers, New York, 1985.

MC68824 Token Bus Controller Preliminary Data Book, Motorola, Inc., Austin, Texas, 1986.

General Motors Manufacturing Automation Protocol—A Communication Network for Open Systems Interconnection, Version 2.1, American National Standards Institute, New York.

Rhonda Alexis Dirvin is the program manager for LAN VLSI in the Motorola 68000 microprocessor operation. Prior to joining Motorola, she attended Cornell University, where she received her BSEE. She is the secretary of the IEEE 802.4 Committee, sits on the MAP Enhanced Performance Architecture Committee, and is a member of the MAP/TOP User Group.

Arthur R. Miller is the guest editor of this issue of *IEEE Micro*. His photo and biography appear on page 7.

Questions about this article can be directed to Rhonda Alexis Dirvin at Motorola, Inc., Mail Stop OE-33, 6501 William Cannon Drive West, Austin, TX 78735-8598.

Multibug: Interactive Debugging in Distributed Systems

Paolo Corsini and Cosimo Antonio Prete
Università di Pisa, Italy

Intended for designers of kernels for distributed multinode systems, this debugger enables them to work in a Unix environment and to interact with the target nodes via a console.

Here, we present an interactive debugger that can be adapted to distributed systems—i.e., *targets*—consisting of several interconnected monoprocessor nodes. The debugger, called Multibug, is intended especially for developers of kernels for distributed systems, but it also offers powerful and flexible support to designers of debuggers for high-level languages. Multibug allows developers to write programs and prepare object files on a host system in a Unix[1] environment, to load object files into a target's nodes, and to interact with all the nodes from a single console via a set of sophisticated commands. Multibug commands can be used in shell programs in such a way that the pair {Unix shell, Multibug commands} is similar to a debugging language.

All Multibug commands utilize constructs similar to those of high-level languages. However, Multibug commands give the user complete visibility of the node architectures. This results from the ability, with Multibug, to assign symbolic names to collections of memory elements, general registers, I/O ports, and so on. Such collections are said to be *variables*, and the commands make it possible to refer to them by name so that inspections and modifications can be carried out. Variables are *typed*, and one can declare new types by starting from five basic structures:

- set_of_bits,
- array_of_sets,
- record_of_sets,
- array_of_records, and
- stack_of_sets.

The type of the variable influences the presentation of its values on the console; typing, therefore, enables the display of process descriptors, protection tables, relocation tables, and so on in a structured and compact way.

0272-1732/86/0600-0026$01.00 © 1986 IEEE

Powerful event management and two flexible commands make the debugging of distributed software possible. At the target level, each node is constructed in such a way that one can

- set breakpoints when local events occur,
- activate an internode (fast) signaling mechanism for generating events in other nodes when a local event has occurred, and
- send event notification messages to the host.

The nature of events is linked to the node architecture, and the internode signaling mechanism is linked to the internode communication support system. The following are typical events:

- the execution of an instruction,
- the modification of data,
- the encountering of a special CPU code (usually a trap or supervisor call), and
- an interrupt, violation, or fault.

Internode interrupt signals (if the target is a multiprocessor) or internode interrupt messages (if the target is a net) are typical internode signaling mechanisms. The Multibug BREAK command allows the developer to specify a subset of the target nodes and a condition on the global target state in such a way that the target nodes stop executing the distributed software that is to be debugged when the condition becomes true. Another command, GO, allows the developer to put programs into execution on one or more nodes, possibly specifying supplementary stop conditions on local events for each node.

We carefully examined the problem of Multibug transportability and solved it by using a host computer running the Unix operating system. Ninety percent of the debugger is housed on the host; this portion is written in the C language[2] and is called HBUG. The remainder of the debugger is a monitor allocated to each node of the target; it executes only simple commands and is called NBUG. NBUG constitutes the only nontransportable code of Multibug. However, the commands it executes are not linked to the node architecture.

Each node is viewed as a *processor* having access to a set of *volumes*. Each volume consists of a set of *segments*, and each segment is a collection of contiguous bytes. Each byte is selectable through an *offset*. Every location in the node is referenced by the triplet {volume, segment, offset}. HBUG, in issuing commands to NBUG, places no restriction on the number and the physical or logical meaning of the volumes and segments. However, memory spaces, I/O spaces, CPU registers, and so on will be typical volumes for each node.

The architecture of the Multibug system is illustrated by Figure 1. The host is connected to the target's nodes via serial point-to-point lines arranged in a star configuration.

When a target has a high number of nodes, a developer may find it useful to implement NBUG in such a way that existing links among the nodes can be used to support the communication between the nodes and the host. The advantage of the star configuration, however, lies in the simplicity and reliability of its connections. Simple and reliable connections ease the debugging of prototype targets, and they reduce the size of NBUG as well as the amount of computing power subtracted from the nodes. Thus, in the following discussion we have stayed with the star configuration.

The host system consists of a general-purpose computer (actually a Zilog System 8000) and a switching board (actually a computer element with a Z8002 processor and 16 serial ports) that efficiently handles low-level communications between the host and the nodes and allows the developer to avoid direct use of the (expensive) serial ports of the host computer.

From the foregoing, one can see that Multibug is intended to fill the gap between simple monitors for uniprocessor boards and debuggers used for high-level languages. The former are oriented to assembler languages and are strictly dependent on the microprocessor architecture,[3-5] whereas the latter completely mask the target's architecture.[6-13] Multibug is also intended to extend debugging facilities to multinode systems[14] and to give the user the opportunity to work in a symbolic way, by freeing him from the "hardware brutalities"[15] without denying him visibility of the target's architecture.

Next, we will briefly describe the protocol by which the host and the target nodes exchange low-level messages. We will then explain the commands that NBUG executes, and

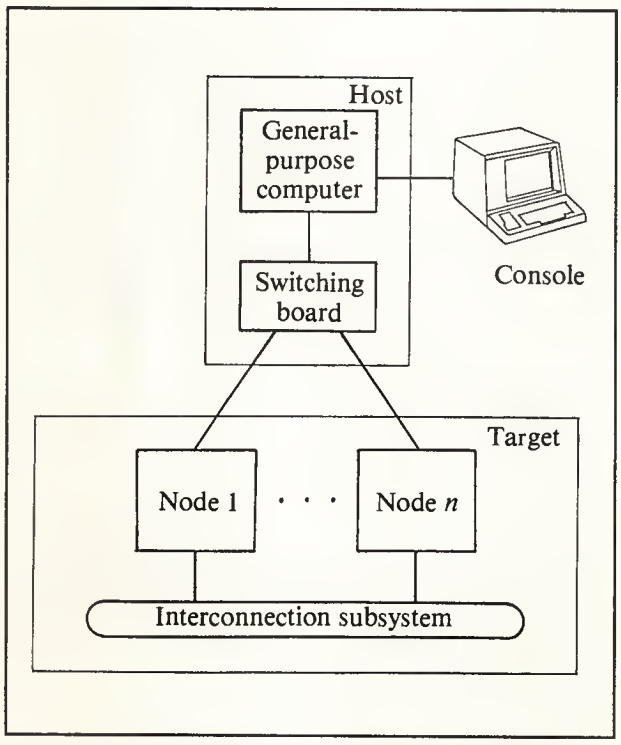


Figure 1. Configuration of the Multibug debugging system.

then discuss those that HBUG executes. Since the HBUG commands are the interface between the developer and the debugger, henceforth we will refer to the HBUG commands as the Multibug commands.

Host-target low-level protocol

The host and each node of the target are connected by a point-to-point serial line. Message exchanges are supported at the low level by a protocol that ensures reliable transmission even in the presence of noise and that prevents deadlock situations resulting from faults in one or more nodes. These features are obtained by having each character sent by the host to a node echoed and by having proper time-outs set both on the wait time for the echo and on the transmission time of the whole message.

An NBUG layer and an HBUG layer (the latter allocated on the switching board) support the low-level protocol. The high-level layers of HBUG communicate only with the lower-level layer of HBUG. More precisely, during a transmission from host to target, higher-level layers of HBUG transfer the pair {body of message to be transmitted, list of names of nodes to which message is to be sent} to the low-level layer. Similarly, during a transmission from target to host, the low-level layer transfers the pair {body of message received, name of node which sent message} to the higher-level layers.

NBUG commands

NBUG can recognize and execute eight commands. After NBUG receives, accepts, and executes a command, it returns an acknowledgment message to HBUG with state information and, possibly, output data. Each command that NBUG executes contains a field that HBUG fills, before it sends the command to NBUG, with a "command unique identifier" derived from the actual value of a command counter. Each acknowledgment message returned by NBUG contains the identifier of the executed command. HBUG uses this information to handle a time-out on the reception of command acknowledgments.

Commands for inspecting and modifying the elements of a volume. The command

DISPLAY volume, segment, offset, nn

returns to HBUG, as output data, the contents of nn consecutive bytes, the starting address of which is specified by the triplet {volume, segment, offset}. The volume field selects one of the spaces accessible in the node; the segment field specifies one segment in the volume and the offset field identifies the first byte in the segment.

The command

MODIFY volume, segment, offset, nn, data

modifies the contents of a block of nn consecutive bytes with the data field, which may be repeated. The first byte of the block is addressed by the triplet {volume, segment, offset}.

Commands for controlling program execution. The command

GO volume, segment, offset

puts a program into execution by taking the triplet {volume, segment, offset} as the starting address.

The command

NEXT nn

provides the single_step capability by putting the program actually pointed to by the program counter register into execution and by returning control to NBUG when nn instructions have been executed.

Finally, the command

HALT

unconditionally stops the program that is actually running and returns control to NBUG.

Commands for managing events. For each node a list of events should be given, and that list should be manageable by NBUG. Each event may be masked or enabled at any time. When an event is masked, the hardware/software event manager ensures that no action is performed. When an event is enabled, the processor state is saved, an event notification message is sent to HBUG, and one or more of the following actions is performed:

- control is returned to the interrupted program,
- control is returned to NBUG,
- an interrupt signal (if the target is a multiprocessor) or an interrupt message (if the target is a net) is sent to other nodes, or
- a user-defined message is inserted into the event notification message.

A code is associated with each event, and at least four event codes should be defined:

- code 0, associated with the detection of the execution of a given instruction,
- code 1, associated with the detection of the modification of data,
- code 2, associated with the detection of an internode interrupt signal or message, and
- code 3, associated with an "exit" directive in the user program.

A mechanism should also be provided to allow the user to define software events according to his personal needs. In NBUG, the user can generate a software event of code j by inserting a special CPU instruction into his user program and leaving code j in a general CPU register.

The command

ENABLE_EVENT eventcode, actions, descriptor

enables an event of code eventcode and specifies the actions to be performed when the event occurs. The descriptor field

is used in the case of a code 0 or code 1 event, and it contains information on the address of the instruction (code 0) or on the address of the data involved in the detection of the event (code 1).

The command

MASK_EVENT flag, eventcode, descriptor

masks one (flag = 1) or all (flag = 0) of the enabled events of code eventcode.

Finally, the command

DISPLAY_EVENT_STATUS

returns the list of all the enabled events to the host.

Multibug commands

Multibug provides the developer with an interactive symbolic command language for debugging software distributed on a target. A Multibug command, which the developer issues from the console, is interpreted and translated into several NBUG commands that are sent to the target nodes. Error messages are sent to the console if a syntax error is detected or if the command itself fails.

Multibug commands can be grouped into several classes:

- commands for describing the target,
- commands for preparing executable files and sending them from the host to the target,
- commands for declaring constants, labels, types, and variables,
- commands for inspecting variables,
- commands for controlling program execution, and
- low-level commands.

In the following, commands are presented in an informal way—key words appear in uppercase letters whereas the parameters the user should specify appear in lowercase letters.

The developer enters Multibug by issuing, under the Unix operating system, the command

MOUNT_MULTIBUG

This command allows him to use not only the standard-shell (or C-shell) Unix commands but also the Multibug commands.

Commands for describing the target. Two commands allow the developer to instruct Multibug on the *configuration* of the target and on the *personalization* of the development environment. By configuration of the target we mean the number of nodes actually connected to the host and, for each node,

- the pair {symbolic name to be assigned to the node, physical address of the host port to which the node is connected},
- the baud rate of the serial line connecting the node to the host,
- the numerical identifiers of the volumes accessible on the node and the symbolic names to be assigned to those volumes, and
- the list of the events that can be managed by the node and the symbolic names of those events.

By personalization of the development environment we mean the set of constants, labels, variable types, and variables the developer has declared.

Multibug provides the developer with an interactive symbolic command language for debugging distributed software.

The command

CONFIGURE file

instructs Multibug that the target configuration is that specified by the referenced file. This command allows the developer to interactively modify the configuration or, if the file is empty, to define it.

The command

PERSONALIZE file

instructs Multibug that the personalization of the development environment is that specified by the referenced file. This file is meaningful to Multibug even if it is empty. The personalization file is modified and updated in a user-transparent way when the commands for declaring constants, labels, types, and variables (see below) are issued.

Command for linking and relocating. The command

```
DO outfile LINKING infile:sec,..., infile:sec
                FROM address
                  AND infile:sec,...
                FROM...
```

allows the developer to link and relocate object modules on the host. It generates two files—outfile.exec and outfile.map. The first is an executable file that can be loaded on the target nodes; the second contains a symbol table and the size of the modules and can be inspected by the developer by means of standard Unix commands. Each input module is specified as infile:sec, where infile is the name of a file having the standard Unix file format a.out and where the parameter sec ϵ (code, data, bss) selects one of the three standard Unix sections (i.e., text, initialized data, or uninitialized data) in the file infile.[16] All the modules listed between the key words LINKING and FROM (or AND and FROM) are relocated consecutively, starting from the specified address. Proper use of the parameter sec gives the developer the ability to split the executable file into

segments (independently relocated) containing only code or only data.

Command for loading. The command

LOAD file ON node AND file ON node AND...

allows the developer to load executable files into the target's nodes. It loads the relevant files into the referenced nodes after determining that the files are of the .exec type.

Commands for declaring constants, labels, types, and variables. The symbols entered into Multibug commands are *labels* and *identifiers* for constants, variables, and types. Labels represent addresses and constant identifiers represent integers. Variable identifiers enable the developer to refer to nonhomogeneous collections (memory and I/O locations, general and state registers, and so on) whose elements are pertinent to the same node. Variables are typed and five basic structures are permitted:

- the set_of_bits (briefly set),
- the array_of_sets,
- the record_of_sets,
- the array_of_records, and
- the stack_of_sets.

Types can be declared and identifiers can be specified in type declarations, allowing the developer to declare variables of that type. Four commands are provided for symbol declaration, and they automatically update the current personalization file.

The command

CONSTANT constant IS_FOR integer

associates the identifier constant with an integer. The integer can be entered in decimal, hexadecimal, or octal format.

The command

LABEL label IS_FOR address

associates the identifier label with an entered address; such an address is a triplet {volume, segment, offset}, each element of which can be entered as an unsigned integer or as a previously declared positive constant identifier.

The command

TYPE type

allows the developer to declare a new type of name type. Multibug interactively asks for all the parameters that characterize the new type.

> Each Multibug command appears to the shell as a standard Unix command.

The command

VAR variable OF type

allows the developer to declare a typed variable of name variable. Multibug interactively asks to which nodes the variable is allocated and also asks for all the parameters it needs so that it can address the variable in each node. Such parameters may be given "directly" by value or "indirectly" as the contents of other variables whose names have already been defined. In the first case, the declared variable is said to be *static*, since its sizes and addresses are statically fixed at the declaration time; in the second case, the variable is said to be *dynamic*, since its sizes and addresses will be computed dynamically by Multibug each time the variable is referenced in a command.

For the sake of brevity, we will not give additional details on variable declarations here, nor will we discuss certain commands able to list and kill constants, labels, types, and variables.

Commands for inspecting and modifying variables. The developer can inspect and modify the contents of a variable with the Multibug commands

DISPLAY variable OF node

and

MODIFY variable OF node

The format in which the contents of the variable are displayed and the modes in which it may be interactively modified are linked to the variable type.

A command

TELL variable OF node

displays the variable's history, that is, the sequence of values that has been assumed and the addresses of the instructions that have modified that variable. The sequence and the addresses are stated in a file generated by the TRACE command, which will be explained later.

Commands for controlling program execution. Four commands allow the developer to control program execution flows, to specify stop conditions for one or more nodes, and to obtain variable traces. They provide the capability for debugging distributed software.[12]

The command

BREAK node,..., node ON condition

sets a breakpoint on all the referenced nodes when the condition becomes true. The condition relates to the state of one or more target nodes (not necessarily those to be stopped) and usually consists of a set of subconditions linked by the OR operator. A subcondition usually consists of a set of partial conditions linked by the AND operator. A partial condition has the form "node:local-condition," in which the local condition is in turn a set of primitive conditions. These primitive conditions are linked by the OR and AND operators and are relevant to the referenced node.

Primitive conditions may be entered as

- symbolic identifiers of events,
- Boolean expressions on variable values, and
- appropriate key words and parameters denoting that the condition is true when nn instructions or a given instruction are executed.

The command

GO node FROM address TILL condition AND
node FROM...

allows the developer to put a program into execution on each of the referenced nodes. He may enter the starting address and a local stop condition for each program. He may omit certain parameters in issuing the command and allow default rules to be applied. These rules are easy to imagine. For example, the command in the form GO ALL restarts program execution on each connected node from the point at which the program had been stopped, and no new stop condition is activated.

The command

HALT node,..., node

immediately stops the referenced nodes. If the command is entered as HALT ALL, it stops all of the target's nodes.

The command

TRACE variable,..., variable ON node
FROM condition1 TO condition2

stores the history of the referenced variables in a file, starting at the point at which condition1 becomes true and ending at the point at which condition2 becomes true.

Low-level Multibug commands. In addition to the commands described above, Multibug provides several low-level commands that are useful in the early stages of testing and debugging the hardware and software of new nodes, and in creating new commands. These commands are

- DUMP node FROM address FOR nn BYTES
- FILL node FROM address WITH data FOR nn TIMES
- SET_EVENT_ON node
- TARGET node

The first two commands are standard ones. The third command allows the developer to activate, deactivate, and list events interactively. The fourth command creates a direct channel between the console and the referenced node, thus putting the host into a transparent state.

The Unix shell and Multibug commands

The Unix shell interpreter offers the user a powerful set of capabilities (such as redirection of standard input and output, linking of commands through pipelines, expansion of metacharacters, and control of program flow) that makes it easier to develop and debug large programs. For these capabilities to be maintained in the Multibug environment, each Multibug command must appear to the shell as a standard Unix command. This feature is set by the initial command, MOUNT_MULTIBUG. This command adds the name of the directory containing the binary file with the Multibug commands to the PATH (or "path" for the C-shell) variable.

Let us show how a shell program can be used to define a new command with the format

DISPLAY_LIST_ON node ALLOCATED_FROM
vol:seg:offset

This command displays a list. We assume that the list consists of several linked elements. Each element contains a 16-bit data field and a 16-bit pointer field. All the elements in the list are allocated to the same volume and the same segment. Thus, the pointer field of each element contains only the offset of the next element on the list. (An offset equal to zero denotes the end of the list itself.)

The structure of the shell program implementing this command is

```
# The character ':' has been defined as a separator.
node = $2;
vol = $4;
seg = $5;
offset = $6
while test $offset != 0000
    do
    set 'DUMP $node FROM $vol:$seg:$offset FOR 4
      BYTES'
    data_field = $1$2
    offset = $3$4
    echo $data_field
    done
```

In this program, the shell command set (in the while loop) executes the Multibug command DUMP and fills the positional parameters $1,$2,$3,$4 with the contents (in hexadecimal ASCII format) of the four bytes inspected by the DUMP command. Next, the contents of the first two of these bytes are assigned to the data_field variable, which is then printed. Finally, the contents of the second two of the bytes are assigned to the offset variable and become the new value.

The command can be implemented more efficiently as a C program, however:

```
# include <ctype.h>
# include <stdio.h>
main(argc,argv)
int argc;
char *argv[ ];
{
  static char type[ ] = 'r';
  char *node,*vol,*seg,*offset,data_field[5],command[36];
  FILE *pipein;
  node = argv[1];
```

```
    vol = argv[3];
    seg = argv[4];
    offset = argv[5];
    while(strcmp(offset,"0000") != 0)
    {
       sprintf(command, "DUMP %s FROM %s:%s:%s
          FOR 4 BYTES", node,vol,seg,offset);
       pipein = popen(command,type);
       fscanf(pipein,"%s %s %s %s",
          &data_field[0],&data_field[2],&offset[0],&offset[2]);
        printf("%s\n",data_field);
       pclose(pipein);
    }
}
```

In this program, the popen statement (in the while loop) creates a process that executes the command previously specified by the sprintf statement as the DUMP command, and sets the pipein variable to point to the file containing the four bytes (in hexadecimal ASCII format) inspected by the DUMP command. Next, the contents of the first two of these bytes are assigned to the data_field variable, which is then printed. Finally, the contents of the second two of the bytes are assigned to the offset variable and become the new value.

Let us give another example of the utility of the Unix shell interpreter—a Unix makefile[17] that handles the DO and LOAD commands in order to automate and speed up the development and the downloading of executable files. We assume that several source files, file1.s, file2.s,..., fileN.s are to be assembled, linked, relocated, and loaded on node 1 of a target as two segments, one containing the code and starting from addresses {00, 00, 0000} and the other containing the data and starting from addresses {00, 01, 0000}.

The text of the makefile, which is self-explanatory, is

```
fileout.ex: file1.o file2.o...fileN.o
  DO fileout.ex LINKING file1.o:code, file2.o:code,...,
  fileN.o:code\
       FROM 00:00:0000\
    AND file1.o:data, file2.o:data, . . . , fileN.o:data\
       FROM 00:01:0000
  LOAD fileout.ex ON 1

file1.o: file1.s
  as -o file1.o file1.s

file2.o: file2.s
  as -o file2.o file2.s
  .
  .
  .

fileN.o: fileN.s
  as -o fileN.o fileN.s
```

We have proposed a debugger, Multibug, suitable for distributed target systems. It allows the user to develop software in a host under Unix operating systems, to load executable modules into the target's nodes, and to perform a debugging session with a set of symbolic and interactive commands. The pair {Unix shell, Multibug commands} is similar to a debugging language. Multibug allows the user to maintain complete visibility of the target architecture.

The user can specify a "personalization" of the development environment by defining typed variables. He can assign a symbolic name and a display format to each variable, according to his needs.

Multibug consists of two modules. The first module, HBUG, runs on the host, is written in the C language, is independent of the target, and constitutes 90 percent of Multibug. The second module, NBUG, resides in each of the target's nodes and "virtualizes" each node's physical structure. The NBUG module is quite simple—it is written for a node with a segmented Z8001 microprocessor and uses only 4K bytes.

Acknowledgments

The work upon which this article was based was supported in part under a grant from the Italian Ministry of Education.

References

1. D. M. Ritchie and K. Thompson, "The Unix Time Sharing System," *Comm. ACM*, Vol. 17, No. 7, July 1974, pp. 365-375.
2. B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, Englewood Cliffs, N.J., 1978.
3. *Z8000 Development Module*, Zilog, Inc., Campbell, Calif., June 1979.
4. *Symbug/A and Debug Monitors Reference Manual*, Pub. no. M68KSYMBG/D3, Motorola Microsystems, Phoenix, Ariz., Aug. 1983.
5. *ICE-86 In-Circuit Emulator Operating Instructions for ISIS-II Users*, Intel Corp., Santa Clara, Calif., 1979.
6. J. D. Johnson and G. W. Kenney, "Implementation Issues for a Source Level Symbolic Debugger," *Proc. ACM Sigsoft/Sigplan Software Engineering Symp. on High-Level Debugging*, Pacific Grove, Calif., Mar. 1983, pp. 173-179.
7. R. E. Fairley, "Ada Debugging and Testing Support Environments," *Sigplan Notices*, Nov. 1980.
8. R. Seidner and N. Tindall, "Interactive Debug Requirements," *Proc. ACM Sigsoft/Sigplan Software Engineering Symp. on High-Level Debugging*, Pacific Grove, Calif., Mar. 1983.
9. B. Beander, "Vax Debug: An Interactive, Symbolic, Multilingual Debugger," *Proc. ACM Sigsoft/Sigplan Software Engineering Symp. on High-Level Debugging*, Pacific Grove, Calif., Mar. 1983.
10. M. Borghesi, P. Mello, and A. Natali, "Il Debugging di Sistemi Multi-Processore Real-Time: Una Proposta di Soluzione Integrata," *AICA Annual Conf. Proc.*, Naples, Italy, Sept. 28-30, 1983, pp. 303-312.
11. E. D. Schiffenbauer, "Interactive Debugging in a Distributed Computational Environment," MS thesis, MIT, Cambridge, Mass., 1981.
12. R. Curtis and L. Wittie, "Bugnet: A Debugging System for Parallel Programming Environments," *Proc. 3rd Int'l Conf. on Distributed Computing Systems*, Oct. 1982, pp. 394-399.
13. M. Boari et al., "Multimicroprocessor Programming Techniques: MML, A New Set of Tools," in *La Linea di Programmazione MML*, Progetto Finalizzato Informatica, Obiettivo Mumicro, Sept. 1983, pp. 15-68.
14. H. Garcia-Molina, F. Germano, and W. H. Kohler, "Debugging a Distributed Computing System," *IEEE Trans. Software Engineering*, Vol. SE-10, No. 2, Mar. 1984, pp. 210-219.
15. B. Elliot, "A High-Level Debugger for PL/I, Fortran, and Basic," *Software—Practice & Experience*, Vol. 12, No. 4, 1982, pp. 331-340.
16. K. Thompson and D. M. Ritchie, *The Unix Programmer's Manual*, Bell Laboratories, 1978.
17. S. I. Feldman, *Make—A Program for Maintaining Computer Programs*, Bell Laboratories, Murray Hill, N.J.

Paolo Corsini is an associate professor of digital computers with the Engineering Faculty of the University of Pisa. His research interests include multi-microprocessor systems and computer architecture. He received the Dott. Ing. degree in electronic engineering, cum laude, from the University of Pisa in 1969.

Questions about this article can be directed to Corsini at the Istituto di Elettronica e Telecomunicazioni, Università di Pisa, Via Diotisalve 2, 56100 Pisa, Italy.

Cosimo Antonio Prete's photo and biography appear on page 42.

A Programmable Debugging Aid for

This specialized hardware assists program debugging and testing and program performance evaluation. It is installed like any other peripheral device.

The increasing cost of software development has led computer designers to consider supporting programming environments in new architectures. Indeed, this interest is only one aspect of a wider problem, that is, the reduction of the so-called semantic gap.[1,2] A central issue is providing the programmer with a set of powerful facilities for program debugging and testing.[3] The requirements for such facilities have been discussed by Goossens,[4] Johnson,[5] and Seidner.[6] A few such requirements can be met easily by a set of debugging aids wholly implemented in software[7,8,9]; however, assistance from the underlying hardware is required whenever the software being developed is real time.[10] A few proposals have been presented that rely on the inclusion of ad hoc hardware in the memory address translation circuitry[4] or even inside the processor.[11-13] These approaches are not attractive, however, if off-the-shelf microprocessors are to be used or if facilities for debugging are intended to be an option to an existing architecture.

We considered the problem of providing hardware support for the development of real-time software in the context of a specific computer architecture, the Selenia MARA family of multi-microprocessor systems, under the constraint that no modification to existing MARA hardware be involved. Our effort led to the specification of a programmable debugging aid, or PDA, that can support program performance evaluation[14,15] as well as program debugging and testing. The PDA hardware has been fully designed. At present, it is being implemented as a research prototype.

The MARA architecture

MARA is a multi-microprocessor architecture developed by Selenia SpA in 1979 and oriented primarily toward real-time process control.[16] A salient feature of the architecture is a high degree of modularity. At the hardware level, this leads to a wide range of physical organizations. A MARA system consists of one or more *nodes* loosely connected via serial links to form a network. The configuration of each node can vary from a single-bus monoprocessor all the way up to a multibus multiprocessor. Moreover, two nodes can be connected in a matched-pair configuration to provide enhanced availability.[17]

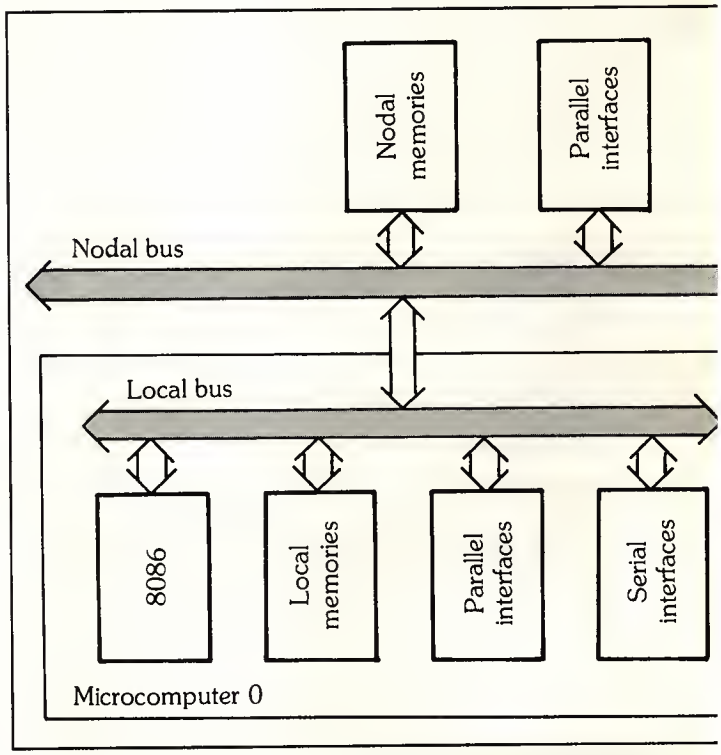


Figure 1. Organization of a MARA multiprocessor node.

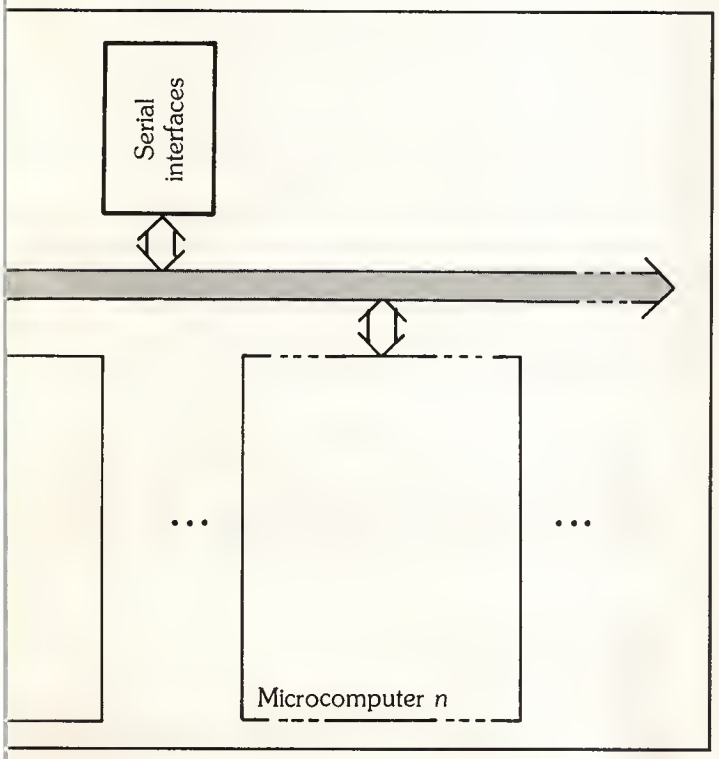
0272-1732/86/0600-0034$01.00 © 1986 IEEE

Real-Time Software Development

Beatrice Lazzerini and Cosimo Antonio Prete
Università di Pisa, Italy

Lanfranco Lopriore
Consiglio Nazionale delle Ricerche, Pisa, Italy

Figure 1 shows the organization of a multiprocessor node. The node consists of up to 16 microcomputers sharing access to a common pool of resources via a *nodal bus*. Besides a nodal memory of up to 512K bytes, shared resources include serial and parallel input/output interfaces. Each interface is fully programmable and includes an Intel 8088 microprocessor, an 8K-byte read/only memory bank, and a 64K-byte read/write memory bank. The serial interface features four serial input/output channels, an 8-bit strobed input port, and an 8-bit strobed output port. The parallel interface supports two independent 512K-byte-per-second parallel input/output channels and is used to connect peripherals such as disks, magnetic tapes, and parallel printers.



Each microcomputer of a node consists of a microprocessor accessing a pool of local resources via a local bus. Local resources include a 512K-byte memory bank and, possibly, serial and parallel interfaces. In the MARA system we used, the microprocessor is an Intel 8086 that can be paired with an Intel 8087 numeric coprocessor. However, the architecture is also designed to host Intel iAPX 286 microprocessors, which allow the size of both the private and the common memory address space to be enlarged to 8M bytes.

The most important features of a node are sophisticated protection mechanisms and heavy hardware supports for error detection. Sixteen protection domains numbered from 0 (the most privileged) to 15 are enforced. They are partitioned into four protection levels. Level 0 includes the single domain 0, whereas levels 1, 2, and 3 include domains 1 and 2, 3 to 5, and 6 to 15, respectively. When a process is running, it works in a specific domain specifying the memory and I/O areas accessible to that process at that time. A six-bit characterization code is associated with each memory page (256 bytes) as well as with each I/O channel (2 bytes). This code specifies the domains having permission to access that page or I/O channel as well as the kind of access permitted. For instance, if domain d is at privilege level p, a memory page whose characterization code states that it is an *interface page* of domain d can be read by that domain as well as by all the domains at a privilege level lower than p, but it can be written by none. A *hierarchic page* of domain d, however, can be read and written by domain d and by all the domains at a privilege level higher than p, and all other domains do not have permission to

access it. A *kernel page* and a *system page* can be read by all the domains; however, a kernel page can be written only by domain 0, and a system page only by domains 0 to 5.

Error detection features of a node include automatic correction of single memory errors, detection of double errors, and detection of parity bits on bus interconnections. Moreover, a *bus timeout* occurs whenever a master of a bus attempts to access a slave unit and obtains no answer within a predefined interval. A *slave error* occurs when a slave unit realizes it is unable to carry out a required action correctly (for instance, as a consequence of an uncorrectable memory error).

Requirements for the PDA

Here, we discuss the major requirements for the PDA, at both the architectural and functional level.

Autonomous processing power. Debugging facilities should not be supported only by PDA hardware—they should also be supported by a host system,[18,26] especially if program performance evaluation and high-level debugging are major requisites.[6,19] But even if host support is provided, the PDA will still have to perform complex tasks such as automatic insertion and deletion of breakpoints (needed to deal with local variables in stack-oriented high-level languages) and event filtering.[14] Thus, the PDA should be provided with autonomous processing power—it should have a set of debugging commands characterized by high granularity implemented on it. (The proper sequences of such commands will be issued by the host system in response to requests from the operator at the debugging console.)

Real-time operation. Interference between the PDA and the processor running the program being tested (the target processor) should be kept to a minimum, not only to ensure correct measurement of execution times but also to allow observation of time dependencies in program behavior. The normal flow of control of the program should not be modified to run the debug routines; execution of these routines should be demanded of the computer element inside the PDA.

Optionality. The PDA is intended to be an optional feature of the MARA architecture, to be plugged in only in installations in which software development will be carried out. We cannot tolerate any modification to the MARA systems, which are built around off-the-shelf microprocessors. Thus, we cannot expect ad hoc debugging facilities to be included in the processor running the program being tested, such as mechanisms for automatic detection of undefined-value errors and value-range violations,[5] trap bits in instruction opcodes,[20] trap registers,[21] and data access matchers for given memory addresses.[13] (We should note, however, that such facilities are worth careful consideration by designers of new microprocessors.)

Large set of hardware-detected events. An *elementary event* is the occurrence of a set of conditions affecting the value of the data item being processed by the program under test as well as the kind of access being performed on that item. An *accumulated event*, on the other hand, is the *n*th occurrence of a particular elementary event.[22] The PDA should include wired logic able to detect a large set of both elementary and accumulated events, so that the computer element will not become overburdened. (Otherwise, the amount of information processing needed to reveal the occurrence of an event via software is likely to be so large that it will slow down the target processor.)

Unmodified object code. A technique commonly adopted for implementing breakpoints is to replace the appropriate instructions in the object code of the program being debugged with as many calls to debug routines.[5,7] However, this technique is viable neither for shared code nor for code stored in read/only memories. (This is the case when we are gathering statistics on the utilization and performance of kernel routines, for instance.) Moreover, the modified memory layout which results from this technique is unacceptable for dealing with position-dependent programming errors.[19]

Architecture of the PDA

The above considerations (especially the requirement for optionality) and the high degree of modularity characterizing the MARA architecture led us to design the PDA as just another programmable interface, one to be connected either to the nodal bus or to a local bus of the target MARA system. A PDA connected to the nodal bus makes it possible to inspect the way the software being tested behaves in relation to the code and the data stored in the nodal memory. A PDA connected to the local bus makes it possible to inspect the contents of the local memory. Of course, one node can be provided with multiple PDAs to interface as many different buses as needed.

The PDA is connected not only to the target bus but also to a host system (possibly a MARA node) via a high-speed serial link with a data rate of about 300K bits per second. This link allows the host system to send debugging commands and receive results that are to be filtered further before being displayed at the operator's console.

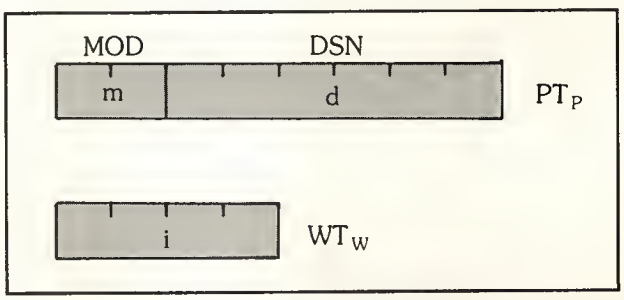


Figure 2. Configuration of the page tag PT_P and the word tag WT_W.

The command set makes it possible to state the set of *conditions* to be detected by the PDA at a particular time as well as the *actions* to be carried out whenever an event is actually detected. The kind of conditions which can actually be revealed by the PDA, and the actions which may possibly be triggered, are described in detail below.

Tags. Let us define a *window* as a 256K-byte memory area starting at a page boundary. At any given time, the PDA connected to a given target bus implements a tagged-memory scheme[2,23-25] on a single window, the *current window*, of the memory address space supported by the bus. This scheme associates two registers (a *condition descriptor* and an *action descriptor*) with each two-byte word of the window. The condition descriptor specifies a set of conditions on the contents of the given word as well as on the kind of access being performed. If these conditions are met, an elementary event (or simply an event) occurs, and the actions specified by the action descriptor are carried out.

Let us examine this process in greater detail. The PDA includes 64 sets of condition descriptors $CD^{(0)}$, $CD^{(1)}$,..., $CD^{(63)}$ and as many sets of action descriptors $AD^{(0)}$, $AD^{(1)}$,..., $AD^{(63)}$. Each set includes 16 descriptors. Let $CD_i{}^{(d)}$ and $AD_i{}^{(d)}$ be the ith descriptors in the dth sets. An eight-bit *page tag* and a four-bit *word tag* are associated with each page and with each word in the current window, respectively. The page tag PT_P pertaining to page P is partitioned into two fields—a two-bit mode (MOD) field and a six-bit descriptor set number (DSN) field (Figure 2).

The contents m of the MOD field specify global conditions on the generation of events in P. More precisely, if $m = (0,0)$, event generation is disabled, whereas if $m = (1,1)$, an event is generated upon every access to a word in P. And if $m = (1,0)$ or $m = (0,1)$, event generation is actually controlled by the contents of the condition descriptors. (It should be clear that this field makes it possible to manage event generation in a whole page efficiently, without having to modify the contents of the tags pertaining to all the words in that page.)

The contents d of the DSN field select the descriptor sets $CD^{(d)}$ and $AD^{(d)}$ pertaining to page P. The specific descriptors $CD_i{}^{(d)}$ and $AD_i{}^{(d)}$, which are inside these descriptor sets and which are actually associated with a given word W in page P, are specified by the contents i of the word tag WT_W pertaining to that word.

Condition descriptors. Figure 3 shows the configuration of the condition descriptor $CD_i{}^{(d)}$. The contents f of the two-bit MOD field specify global conditions on the meaning of the other fields of the descriptor. These fields are meaningful only if $f = (0,1)$ or $f = (1,0)$. If $f = (0,0)$, event generation is disabled for W, and if $f = (1,1)$, an event occurs corresponding to every access to W. The one-bit read (RD) and write (WR) fields, if set, enable the generation of an event corresponding to a read or write access to W. The contents s of the MSK field represent a mask for the contents w of word W. The quantity $w^* = w \wedge s$ is compared with both the contents l of the 16-bit lower-bound (LB) field and the contents u of the 16-bit upper-bound (UB) field, and an event is generated according to the result of an in-range or out-of-range comparison as selected by the one-bit IR/OR field. (In both cases, the boundary values are included in the range.)

Being able to select between read and write accesses may be useful in detecting erroneous attempts to read the values of uninitialized variables or write into code areas. The MSK field makes it possible to deal with a data item smaller than a whole word, such as a short integer implemented in a single byte or a Boolean supported by a single bit. (Indeed, this field even allows us to check a specific condition on a whole subset of the elements of an array of Booleans, provided that all the elements are implemented in the same word.)

An example of the use of bound comparison is the forcing of range checks on the values of variables of scalar types, as well as on array subscripts, whenever such checks are not inserted by the compiler in the object code. It is even possible to detect when a variable assumes a specific value by assigning this value to both the LB and UB fields. Single-bound conditions may be specified, too, by simply setting the other bound to its limit value.

As far as the MOD field of a page descriptor is concerned, it should be clear that the global control it enforces on event generation applies only to the words associated with the descriptor. In contrast, the control the MOD field of a page tag enforces concerns all the words in the page.

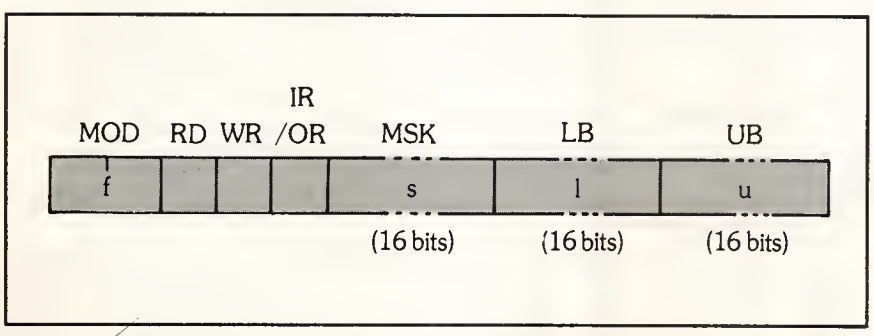


Figure 3. Configuration of the condition descriptor $CD_i{}^{(d)}$.

Action descriptors. The hardware resources of a PDA include a computer element CE, a 128-cell FIFO queue (each cell of which fills an entire state of a MARA bus), and a set of seven counters/timers CT_1, CT_2,...,CT_7. These are the resources involved in the actions caused by an event concerning word W (and specified by the action descriptor $AD_i^{(d)}$ pertaining to that word). Figure 4 shows the configuration of $AD_i^{(d)}$. The one-bit local interrupt request

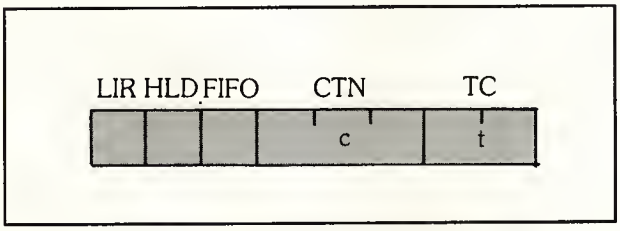


Figure 4. Configuration of the action descriptor $AD_i^{(d)}$.

(LIR) field, if set, specifies that an interrupt request must be sent to the computer element. The one-bit hold (HLD) field, if set, specifies that the processor currently mastering the target bus must be put into a hold state. The one-bit FIFO field, if set, specifies that the current configuration of the target bus must be saved in the FIFO queue. The contents c of the three-bit counter/timer number (CTN) field specify the counter/timer CT_c involved in the event ($c = 0$ means that no such device is actually involved). If CT_c has been programmed by the computer element to work as a counter, it decrements its own internal value and, upon reaching value 0, generates a local interrupt request to the computer element itself. If CT_c has been programmed to work as a timer, it performs interval measurement with an accuracy of 1 μs by either starting, stopping, or switching between the start and stop states, according to the contents t of the two-bit timer control (TC) field.

The information gathered in the FIFO queue is used by the computer element for delayed analysis of the target state causing the event to occur. This makes it possible to collect target states in real time. Indeed, the LIR field is intended to be set only for those words in which (1) a data or code breakpoint requiring the attention of the operator is implemented, or (2) preprocessing of the information gathered in the FIFO queue before it is sent to the host system, is required, or (3) management of internal PDA resources—to set and reset tags at the beginning and at the end of the execution of a subprogram, for instance—is implied. Counters can be used to compute the number of occurrences of a given elementary event; moreover, they allow us to detect accumulated events in an efficient way, without the intervention of the computer element. Timers enable us to measure program execution times. For example, if d1 represents the contents of the DSN field of the tag pertaining to the memory page storing the first instruction of a given routine, and d2 represents the contents of the DSN field of the tag pertaining to the memory page storing the last instruction, the execution time of the routine can be measured by

- tagging the storage units corresponding to the first and last instruction with two different word tag values, say t1 and t2,
- setting $CD_{t1}^{(d1)}$ and $CD_{t2}^{(d2)}$ to detect read accesses,* and
- setting $AD_{t1}^{(d1)}$ to start, and $AD_{t2}^{(d2)}$ to stop, the same timer CT_c*.

Implementing the PDA

Logical configuration. Figure 5 shows the PDA hardware configuration. The computer element CE consists of an Intel 8088 microprocessor, a 16K-byte read/only memory bank, and a 64K-byte read/write memory bank. The read/only memories contain the firmware implementing the debugging commands, which are received from the host via a serial I/O channel. Command execution is supported by the read/write memories, where the debug databases are stored. The target interface logic, or TIL, connects the PDA to the target bus, be it local or nodal. The computer element must be allowed to access the memory bank on this bus so that it can inspect the values of the data items pertaining to the program being debugged. Thus, the PDA is a master for the target bus. The TIL therefore includes all the logic needed for distributed bus arbitration. However, the main function of the TIL is to translate the addresses generated by the 8088 into the format required by the target bus. The TIL includes an address translation register (ATR), whose contents specify the base of a 256K-byte portion of the memory space implemented on the target bus. (As will be clarified shortly, this portion may not be the same as the one which forms the current window.) Addresses generated by the 8088 in the range from (10000)H to (4FFFF)H are relocated by the TIL in this memory portion, whereas addresses ranging from (FC000)H to (FFFFF)H and from (00000)H to (0FFFF)H are processed inside the CE in the read/only and read/write memory banks, respectively. Moreover, the TIL makes it possible to send interrupt requests to the target bus and to force the target processor into the hold state. Let us examine this in greater detail. An interrupt request can be generated only by the 8088 and is aimed at gaining the attention of the target processor so that the contents of the target's internal registers can be inspected after a breakpoint has been reached. A request for the target processor to enter the hold state can be generated only by the event detecting and

*The contents l and u of the LB and UB fields of each descriptor must be equal to (0000)H and (FFFF)H, respectively. In this way, no effective bound check will actually be carried out on the contents of the storage units being tagged, i.e., on the kind of instruction stored therein.

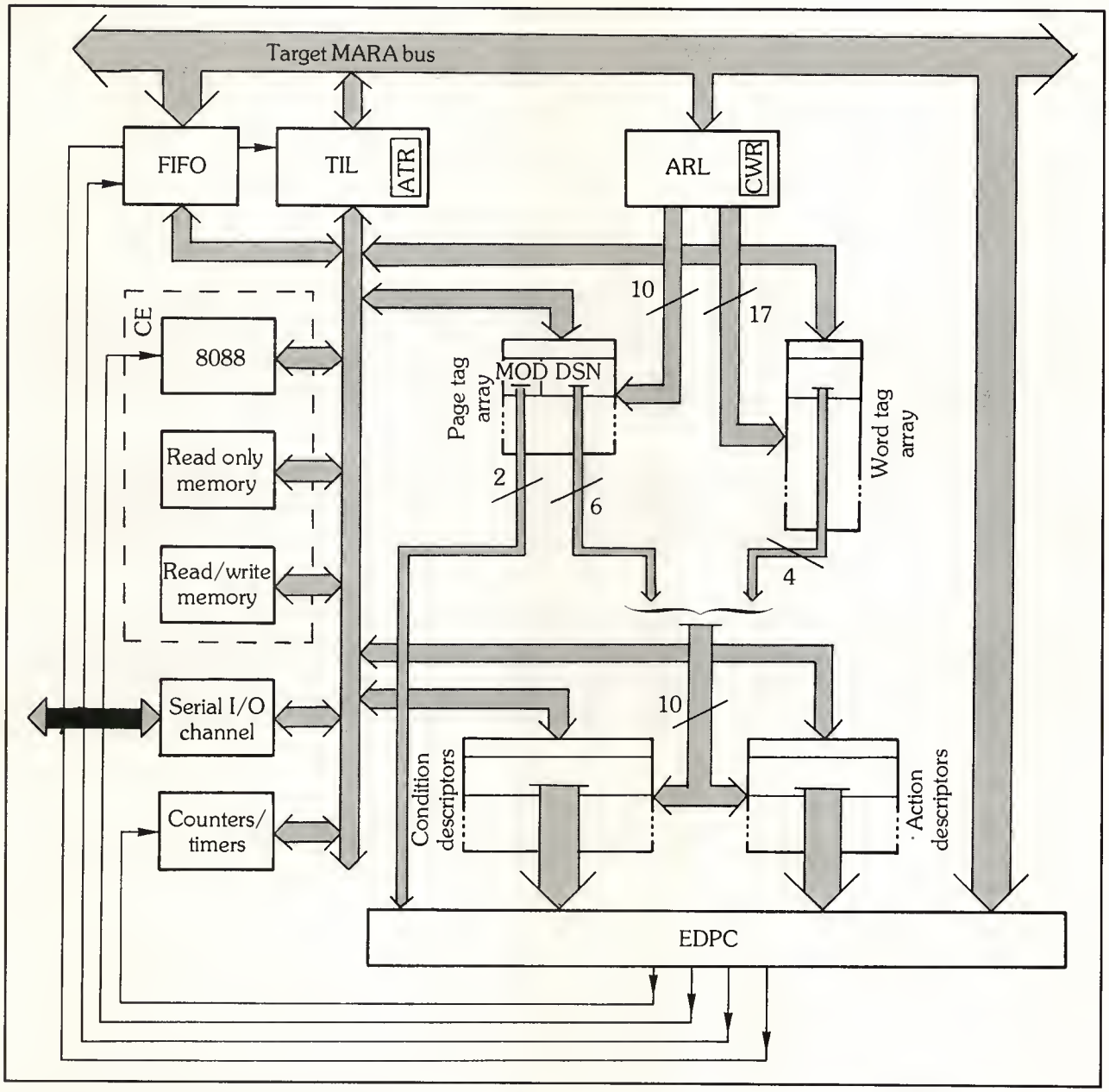


Figure 5. PDA hardware configuration.

processing circuitry (EDPC) after the occurrence of an event, and it can be removed only by direct intervention of the 8088.

A MARA address (either on the nodal bus or on a local bus) consists of 24 bits. Since a window starts at a page boundary, the window of a word W currently being addressed on the target bus is specified by the 16 most significant bits of the address (Figure 6). The address relocation

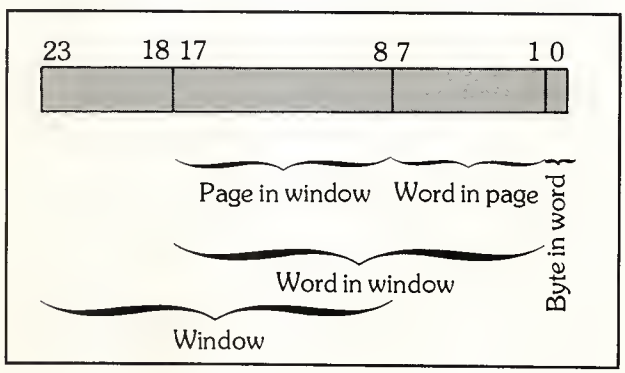


Figure 6. A MARA address, as received by the PDA from the target bus.

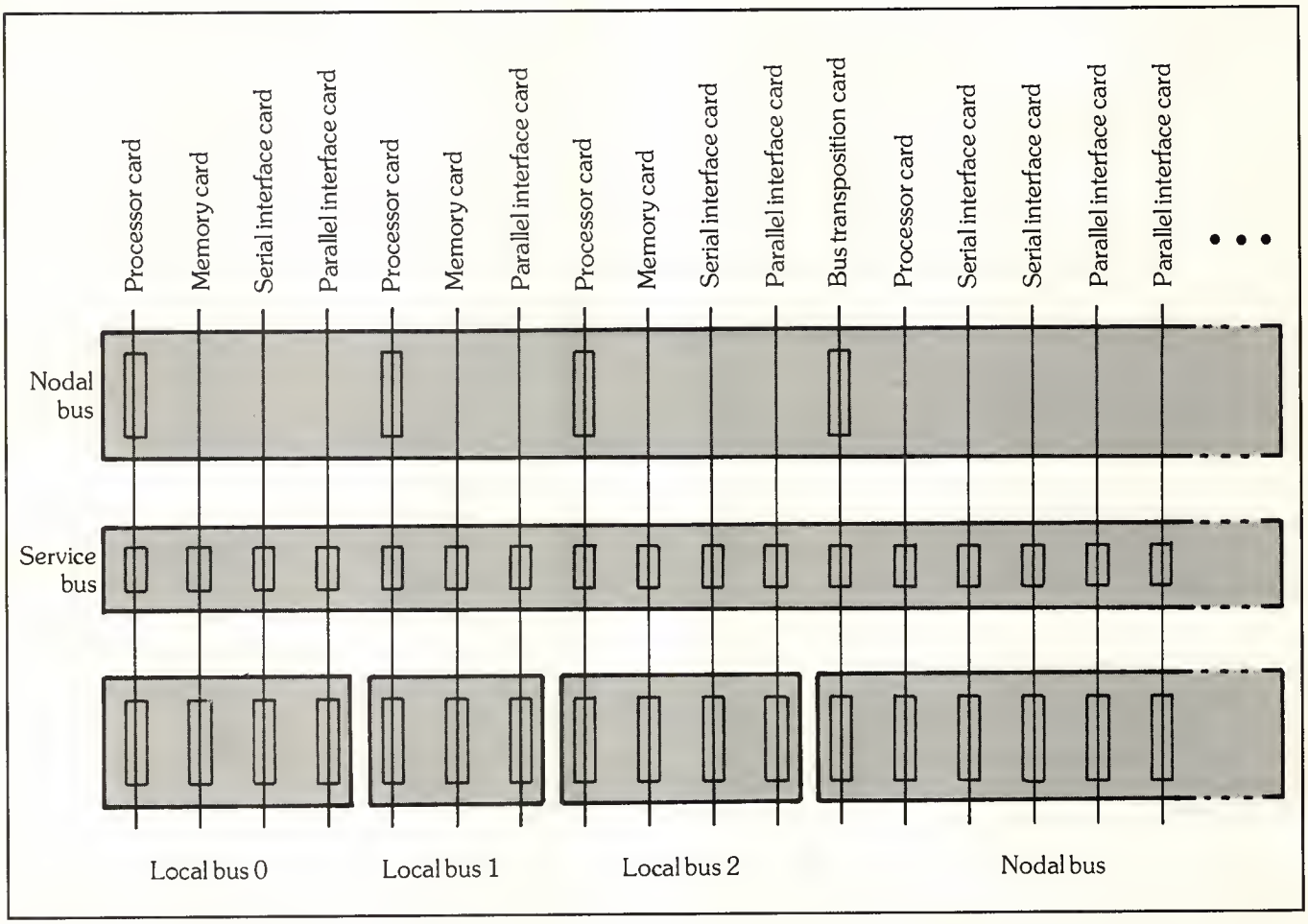


Figure 7. Physical configuration of a MARA system backplane—this configuration features three microcomputers.

logic ARL compares these bits with the contents of a 16-bit current window register CWR. If a match occurs, bits 8 to 17 of the address specify the page P pertaining to word W.

These bits are sent to the page tag array (consisting of 1024 page tags) to select the tag PT_P. Bits 1 to 17 specify the word address inside the current window and are therefore sent to the word tag array (consisting of 2^{17} word tags) to select the tags WT_W. The six-bit contents of the DSN field of PT_P are paired with the four-bit contents of WT_W, and the resulting quantity is sent to both the condition descriptor array and the action descriptor array (each consisting of 1024 descriptors) to select the condition descriptor $CD_i^{(d)}$ and the action descriptor $AD_i^{(d)}$ for word W. The contents of $CD_i^{(d)}$ are sent in turn to the event detecting and processing circuitry EDPC, which compares them with the present state of the target bus. If the occurrence of an event is revealed, the EDPC—depending on the contents of $AD_i^{(d)}$—generates either an interrupt request to the 8088 or the appropriate control signals to the TIL (to force the target processor into the hold state), the FIFO queue (to cause storage of the entire state of the target bus), and a counter/timer.

Physical configuration. A MARA backplane is partitioned into a nodal bus, a service bus for power supplies, and local buses (Figure 7). Each processor card implements a link between its own local bus and the nodal bus and therefore has three connections to the backplane. Memory and I/O interface cards, on the other hand, are connected to the service bus and a local bus only. However, these cards may implement nodal resources as well. For this purpose, the nodal bus may be transposed into the lower position of the backplane by means of an ad hoc bus transposition card linking the upper connector to the lower connector.

A PDA should be connected either to the nodal bus or to a local bus, according to the physical localization of the code or data to be inspected. The ability to transpose buses makes it possible to use the same PDA card type in both

cases. A PDA card features not only two connections to the backplane but also an additional connection in the front for a serial link to the host system.

We feel that we have met the design goals stated earlier in this article. Our PDA is truly an optional feature—it can be installed in any existing MARA system in the same way as any other peripheral device. Real-time operation is supported not only by the computer element but also by other hardware resources inside the PDA, such as the FIFO queue for gathering target bus states and the counters/timers for detecting accumulated events and measuring time intervals. Condition descriptors make it possible to specify sophisticated event conditions; together with page tags and word tags, they represent a valid alternative to putting ad hoc facilities inside the target processor. Furthermore, tags allow a straightforward implementation of code and data breakpoints that does not involve alteration of the object code of the program being debugged.

In the very near future, computer architects will start paying more attention to hardware facilities for program debugging and testing and program performance evaluation. We hope our effort will make a useful contribution to future developments.

Acknowledgments

The work described here was supported by a research contract between Selenia SpA, Rome, Italy, and the Consiglio Nazionale delle Ricerche (National Council of Research), Pisa, Italy.

References

1. U. O. Gagliardi, "Report of Workshop 4—Software Related Advances in Computer Hardware," *Proc. Symp. on the High Cost of Software*, Monterey, Calif., 1973, pp. 99-119.
2. G. J. Myers, *Advances in Computer Architecture*, 2nd ed., Wiley-Interscience, New York, 1982.
3. A. R. Brown and W. A. Sampson, *Program Debugging*, Macdonald/American Elsevier, 1973.
4. M. Goossens and J. Tiberghien, "High Level Debugging Aids for Real Time Software," *Proc. Euromicro Symp.*, Madrid, Spain, Sept. 1983, pp. 71-78.
5. M. S. Johnson, "Some Requirements for Architectural Support of Software Debugging," *Proc. Symp. on Architectural Support for Programming Languages and Operating Systems*, Palo Alto, Calif., Mar. 1982, in *Computer Architecture News*, Vol. 10, No. 2, Mar. 1982, *Sigplan Notices*, Vol. 17, No. 4, Apr. 1982, pp. 140-148.
6. R. Seidner and N. Tindall, "Interactive Debug Requirements," *Proc. ACM Sigsoft/Sigplan Software Engineering Symp. on High Level Debugging*, Pacific Grove, Calif., Mar. 1983, in *Software Engineering Notes*, Vol. 8, No. 4, Aug. 1983, *Sigplan Notices*, Vol. 18, No. 8, Aug. 1983, pp. 9-22.
7. B. Beander, "VAX DEBUG: An Interactive, Symbolic, Multilingual Debugger," *Proc. ACM Sigsoft/Sigplan Software Engineering Symp. on High Level Debugging*, Pacific Grove, Calif., Mar. 1983, in *Software Engineering Notes*, Vol. 8, No. 4, Aug. 1983, *Sigplan Notices*, Vol. 18, No. 8, Aug. 1983, pp. 173-179.
8. B. Elliot, "A High-Level Debugger for PL/I, Fortran and Basic," *Software—Practice & Experience*, Vol. 12, No. 4, 1982, pp. 331-340.
9. J. J. Hart, "The Advanced Interactive Debugging System (AIDS)," *ACM Sigplan Notices*, Vol. 14, No. 12, Dec. 1979, pp. 110-121.
10. R. L. Glass, "Real-Time: The 'Lost World' of Software Debugging and Testing," *Comm. ACM*, Vol. 23, No. 5, May 1980, pp. 264-271.
11. G. W. Gerrity, "Hardware Detection of Undefined References," *Computer Architecture News*, Vol. 8, No. 2, Mar. 1980, pp. 8-11.
12. D. D. Hill, "A Hardware Mechanism for Supporting Range Checks," *Computer Architecture News*, Vol. 9, No. 4, June 1981, pp. 15-21.
13. R. E. McLear, D. M. Scheibelhut, and E. Tammaru, "Guidelines for Creating a Debuggable Processor," *Proc. Symp. on Architectural Support for Programming Languages and Operating Systems*, Palo Alto, Calif., Mar. 1982, in *Computer Architecture News*, Vol. 10, No. 2, Mar. 1982, *Sigplan Notices*, Vol. 17, No. 4, Apr. 1982, pp. 100-106.
14. D. Ferrari, *Computer Systems Performance Evaluation*, Prentice-Hall, Englewood Cliffs, N.J., 1978.
15. S. H. Fuller and P. N. Oleinick, "Initial Measurements of Parallel Programs on a Multi-Mini-Processor," *Proc. Compcon Fall 76*, Washington, D.C., Sept. 1976, pp. 358-363.
16. I. R. Martin, "MARA: An Overview of the Civil Implementation," Selenia Internal Report SPM 2.1, Dec. 1981.
17. P. Ciompi et al., "A Highly Available Multimicroprocessor System for Real-Time Applications," *Proc. Third IFAC/IFIP Workshop (Safecomp 83)*, Cambridge, U.K., Sept. 1983, pp. 247-253.

18. C. R. Hill, "A Real-Time Microprocessor Debugging Technique," *Proc. ACM Sigsoft/Sigplan Software Engineering Symp. on High Level Debugging,* Pacific Grove, California., Mar. 1983, in *Software Engineering Notes*, Vol. 8, No. 4, Aug. 1983, *Sigplan Notices*, Vol. 18, No. 8, Aug. 1983, pp. 145-148.

19. J. D. Johnson and G. W. Kenney, "Implementation Issues for a Source Level Symbolic Debugger," *Proc. ACM Sigsoft/Sigplan Software Engineering Symp. on High Level Debugging*, Pacific Grove, Calif., Mar. 1983, in *Software Engineering Notes*, Vol. 8, No. 4, Aug. 1983, *Sigplan Notices*, Vol. 18, No. 8, Aug. 1983, pp. 149-151.

20. M. D. Shapiro, "The Criterion Cobol System," *Proc. AFIPS Conf.*, Vol. 47, 1978 NCC, pp. 1049-1054.

21. R. P. Case and A. Padegs, "Architecture of the IBM System/370," *Comm. ACM*, Vol. 21, No. 1, Jan. 1978, pp. 73-96.

22. S. H. Fuller, R. J. Swan, and W. A. Wulf, "The Instrumentation of C.mmp, a Multi(Mini)Processor," *Proc. Compcon 73*, San Francisco, Calif., Feb. 1973, pp. 173-176.

23. E. A. Feustel, "On the Advantages of Tagged Architecture," *IEEE Trans. Computers*, Vol. C-22, No. 7, July 1973, pp. 644-656.

24. J. K. Iliffe, *Basic Machine Principles,* Macdonald, 1968.

25. L. Lopriore, "Capability-based Tagged Architectures," *IEEE Trans. Computers*, Vol. C-33, No. 9, Sept. 1984, pp. 786-803.

26. J. H. Hughes, "DIAMOND—A Digital Analyzer and Monitoring Device," *Proc. Seventh IFIP WG 73 Int'l Symp. on Computer Performance Modeling, Measurement, and Evaluation (Performance 80)*, Toronto, Ontario, May 1980, in *Performance Evaluation Review,* Vol. 9, No. 2, Summer 1980, pp. 27-34.

Beatrice Lazzerini is a researcher at the Instituto di Elettronica e Telecomunicazioni of the Università di Pisa, Italy. Her interests include logic and functional programming languages and program debugging and testing. She received the doctoral degree in computer science from the Università di Pisa in 1981.

Cosimo Antonio Prete has performed postgraduate research in computer architecture at Selenia SpA, Rome, Italy. At present, he is with the Instituto di Elettronica e Telecomunicazioni of the Università di Pisa. His interests include multi-microprocessor organization and software development methodologies. He received the doctor of engineering degree in electronic engineering from the Università di Pisa in 1982.

Lanfranco Lopriore is a researcher at the Instituto di Elaborazione delle Informazione of the Consiglio Nazionale delle Ricerche, Pisa, Italy. He has researched memory structures in capability environments, capability-based tagged architectures, and cache memories. He is currently working on hardware tools for program debugging and program performance evaluation.

Lopriore received the doctor of engineering degree in electronic engineering in 1978 and an advanced degree in computer science in 1980, both from the Università di Pisa.

Questions about this article can be directed to Lanfranco Lopriore, Istituto di Elaborazione delle Informazione, Consiglio Nazionale delle Ricerche, Via Santa Maria 46, 56100 Pisa, Italy.

An Intelligent Braille Display Device

Clifford P. Grossner, Thiruvengadam Radhakrishnan, and Alex Schena
Concordia University

The high cost of braille output devices has prevented their wide use among the visually handicapped. Here, a much cheaper device is implemented.

A basic need of every visually handicapped person is the need to communicate with the outside world, with sighted as well as with other visually handicapped persons. Moreover, if a visually handicapped person accepts work where computers are used, he needs to be able to communicate with them. Man-to-computer communication takes place mostly through a keyboard. Because many visually handicapped persons are trained to use a keyboard, they do not have great difficulties with input to a computer. However, presenting computer output to the visually handicapped in a form they can understand requires unconventional output devices such as speech synthesizers and braille displays. These devices may be used individually or jointly to provide effective transfer of information from a computer to a visually handicapped person.[1]

Here we describe the development of an intelligent braille display device that plugs into the standard serial port of any personal computer. The device is controlled by a dedicated single-chip microcomputer, the Zilog Z8671. The personal computer sends the device commands and ASCII text which the Z8671 interprets for display in braille format. The device is line-oriented, and in our prototype one line consists of 40 characters. Each character is displayed by means of a "braille cell" in which there are six dot positions in a matrix of three rows by two columns (Figure 1). A dot is displayed by the raising of a pin in one of the dot positions. Among visually handicapped persons, this type of braille display device is known as "paperless braille."[2]

The visually handicapped and personal computers

The proliferation of personal computers has made them attractive to the visually handicapped for many uses. Personal computers can give the visually handicapped access to databanks (videotex) and electronic mail; they can provide them with improved opportunities to work as computer programmers, data entry operators, word processing operators, reservation clerks, and voice message handlers.

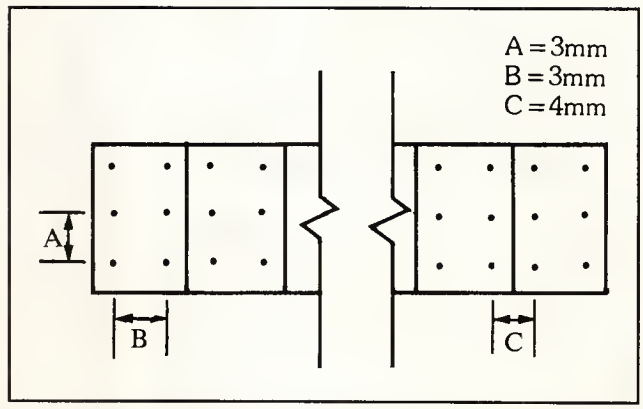


Figure 1. Braille display bar. Note that a character is displayed in a 3 × 2 braille cell.

0272-1732/86/0600-0043$01.00 © 1986 IEEE

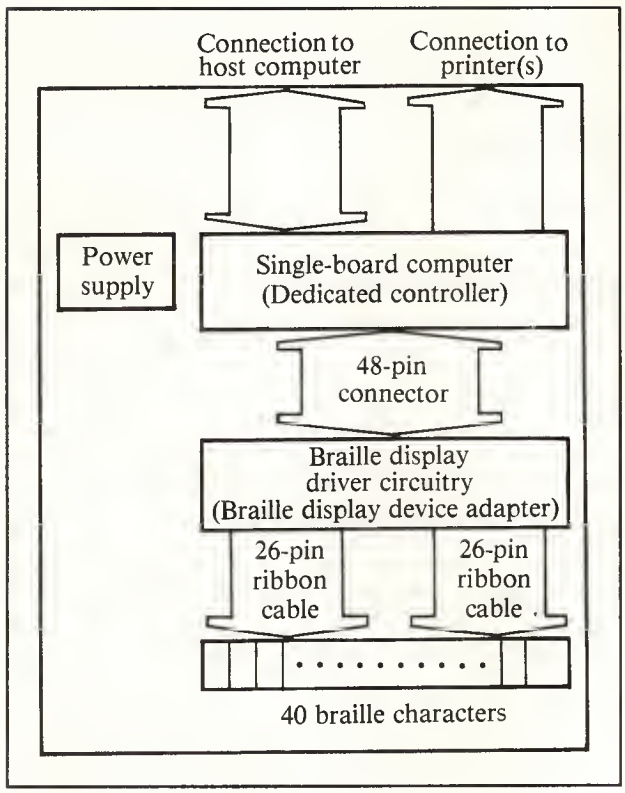


Figure 2. The braille display device.

To use a computer, a visually handicapped person will require one or more of the following types of computer output:

- speech output (to augment the braille output or help those who cannot read braille),
- paperless braille output,
- braille printed output (for the user's files),
- regular printed output (so the visually handicapped user can communicate with sighted coworkers), and an
- enlarged or magnified character display (for the partially sighted).

Each type of output has advantages and disadvantages. Speech is omnipresent and can be heard simultaneously by many people. Speech leaves the listener's hands free to do other things.[3] However, although speech can be stored on a magnetic medium, such a medium does not easily permit the random accessing of speech segments. Soft-copy or hard-copy braille output can be read, reread, and assimilated at a much faster rate than speech. However, those visually handicapped persons whose fingertips are not sensitive enough cannot read braille.

Generalized text-to-speech conversion systems are commercially available at low cost.[4] They can be plugged into the standard serial port of any personal computer. Most of them are based on Vortrax's SC-01 speech synthesizer chip,[5] a CMOS circuit containing an electronic model of the vocal tract. Using this internal model and input codes

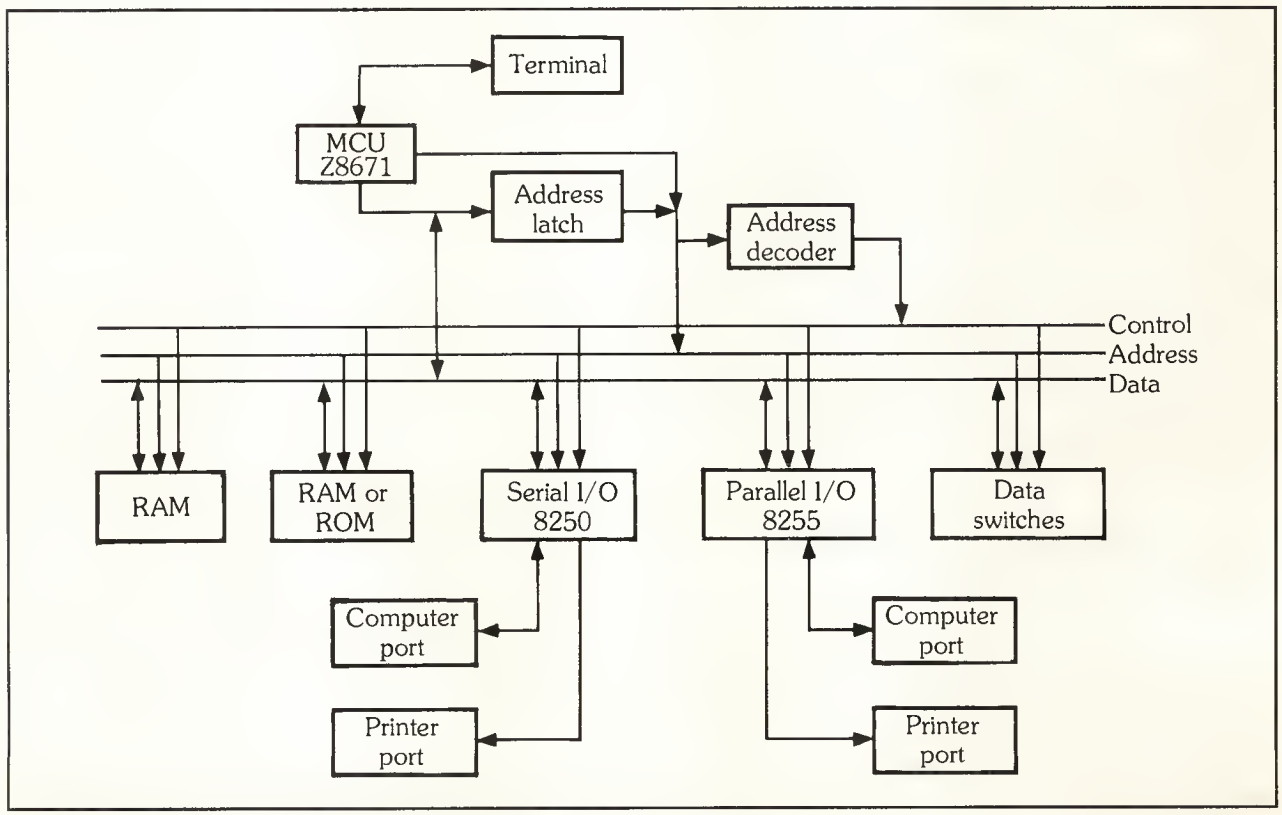


Figure 3. The digital logic of the braille display device's SBC.

representing the phonemes of the English language, the SC-01 produces synthesized voice.

Unlike speech synthesizers—which are electronic, compact, and inexpensive—braille display devices are electromechanical and costly. The popular and commercially available Versabraille system, which contains a 20-character braille display, magnetic tape for mass storage, and a braille keyboard,[2] costs well over $5000. Our braille display was designed to reduce the high cost of such devices.

Functional description

The electronics part of our braille display device consists of two separate sections (Figure 2). The "dedicated controller" is a single-board computer (SBC) based on the Z8671 microcomputer. Figure 3 shows the major components of the board. The "braille display device adapter" consists of the electronics, both digital and analog, the SBC needs to control the mechanical portion of the braille display. In effect, the braille display device adapter allows the SBC to select one of the 40 cells of the braille display as well as the pins of the cell that are to be written or erased.

The SBC is responsible for controlling the operation of the braille display. It receives ASCII text from the personal computer (host), converts the text into braille, and displays it line by line on the braille display unit. The user may direct the braille translation to be in Grade I, Grade II, or computer braille. Grade I braille consists of the standard alphabet, punctuation signs, and a set of composite signs used to indicate capital letters, numbers, and so on. Grade II braille is a superset of Grade I braille and includes a set of contractions that are symbolic codes, or n-grams, for frequently occurring words.[6] Computer braille allows the display of the full range of ASCII characters by using two braille cells per character as control characters.

The braille display is line-oriented. The host computer sends commands to the SBC that allow it to control the operating modes of the braille display. The SBC also relays status information to the host computer that enables the host and the SBC to be synchronized as each line is displayed. This synchronization is necessary because the number of characters in the translated braille line is not necessarily equal to the number of characters in the original line of ASCII text. The command codes accepted by the SBC are shown in Table 1.

Table 1.
Command codes accepted by the SBC.

Mnemonic	Operating mode	Description
P	Enable/disable printer	Indicates that data are to be sent to the printer port as well as to the braille display (toggle).
D	Enable/disable braille display	Indicates that data are to be sent to the braille display as well as to the printer (toggle).
/	Parallel printer	Indicates that a parallel printer is currently attached to the SBC.
S	Serial printer	Indicates that a serial printer is currently attached to the SBC.
1	Grade I braille	Translates all incoming data into Grade I braille.
2	Grade II braille	Translates all incoming data into Grade II braille.
C	Computer braille	Translates all incoming data into computer braille.
T	Truncate	Truncates data lines when they are longer than 40 braille characters.
E	Enquire	Requests status of the current line. This causes the SBC to indicate to the host if the entire current line was displayed in 40 characters.
N	Next	Displays the next portion of the current line.
B	Break line at word/character	When a line, after translation, is longer than 40 characters, splits the line at a word/character boundary (toggle).

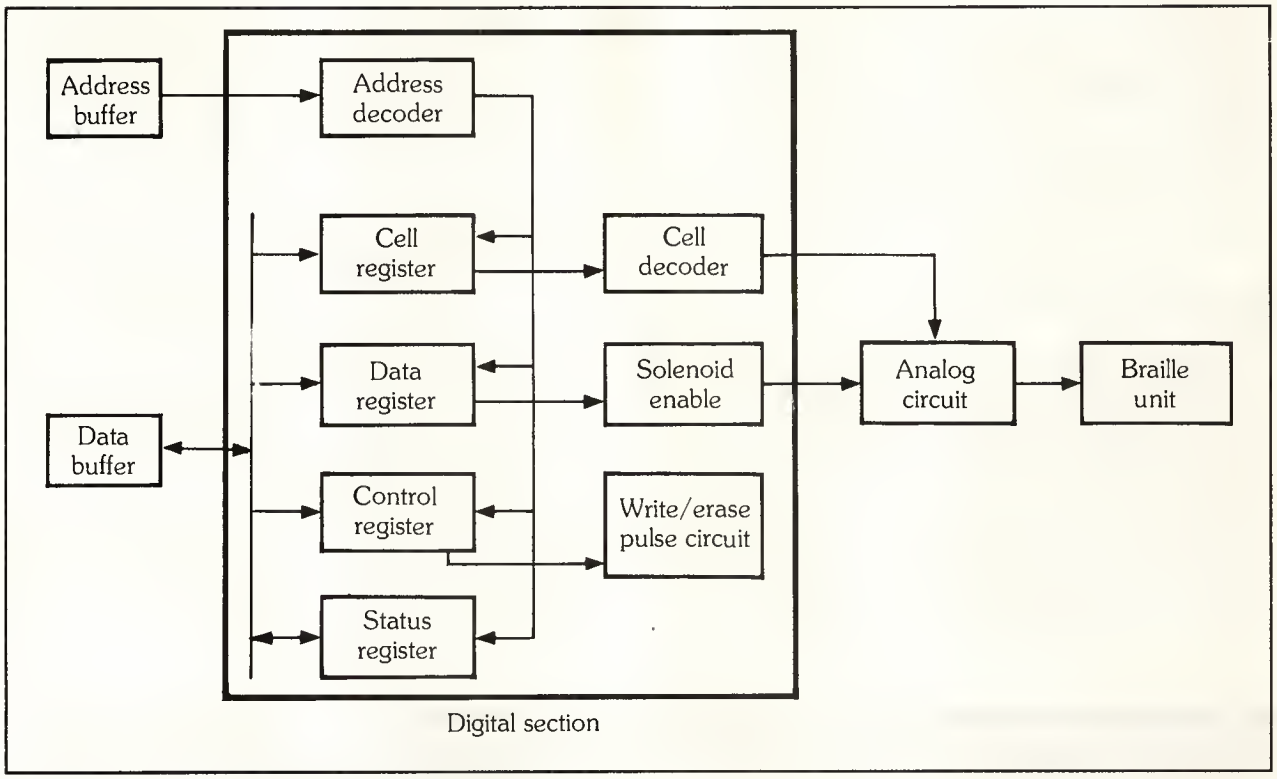


Figure 4. The braille display device adapter.

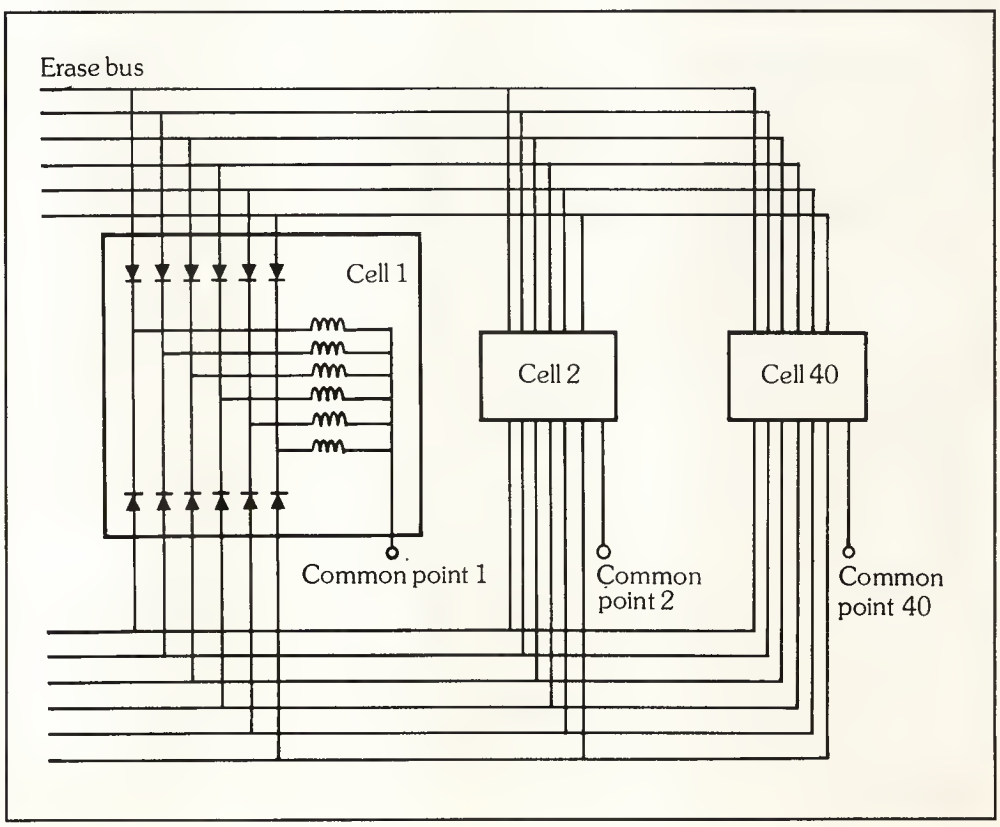


Figure 5. Braille cell schematic diagram.

The SBC provides both serial and parallel communications to the host computer and the printers that can be attached to it. As can be seen in Figure 2, the connection to the host is bidirectional whereas that to the printers is output only. By attaching a dot-matrix printer to the serial port and a braille printer to the parallel port, the user can obtain printed and brailled output simultaneously.

Braille display device adapter. The SBC uses the device adapter to generate the signals to drive the braille display's transducers. The adapter's digital electronics are responsible for

- selection of one of the braille cells for writing or erasing,
- selection of one or more of the six pins within a cell to participate in the write or erase operation,
- generation of interrupts to indicate completion of the write or erase operation, and
- generation of the timing pulses that control the write or erase cycle.

The SBC interfaces with the device adapter through four registers—the cell, data, control, and status registers (Figure 4). The SBC can address these registers in its off-board memory space. The cell register contains the number of the braille cell to be written or erased. In our prototype, any number in the range 0 to 39 is valid. The low-order six bits of the data register contain a bit pattern corresponding to the braille character to be displayed on the cell indicated by the contents of the cell register. The control register is used to choose a write or erase operation as well as to enable or disable interrupts to the SBC. The two low-order bits of the status register indicate if the write/erase cycle is currently in progress.

Table 2.
Characteristics of the braille transducer.

	Write	Erase
Voltage (min.)	12 V	9 V
Current	193 mA	145 mA
Power dissipation	2.31 W	1.3 W
Energy	23 mWs	13 mWs
Pulse polarity	−	+
Pulse period	10 ms	10 ms
Waveform	⊓	⊔

A braille cell consists of six miniaturized push-pull solenoids whose plungers can lock in an up or down position. The movement of each plunger is controlled by the direction of the current flow through its solenoid. A positive current flow—that is, from the erase bus to a common point (Figure 5)—causes the plunger to be lowered; this is an erase operation. A negative current flow—that is, from the common point to the write bus—raises the plunger; this is a write operation. Table 2 gives the characteristics of the braille transducer. This transducer is available from the Tiflotel company in modules of one, eight, 20, or 40 cells.[7]

The circuit shown in Figure 6a is used to write or erase one pin of a braille cell. The circuit, except for the triac T1, is replicated six times, once for each solenoid in the cell. When Q1 conducts, +12V is applied at one end of the

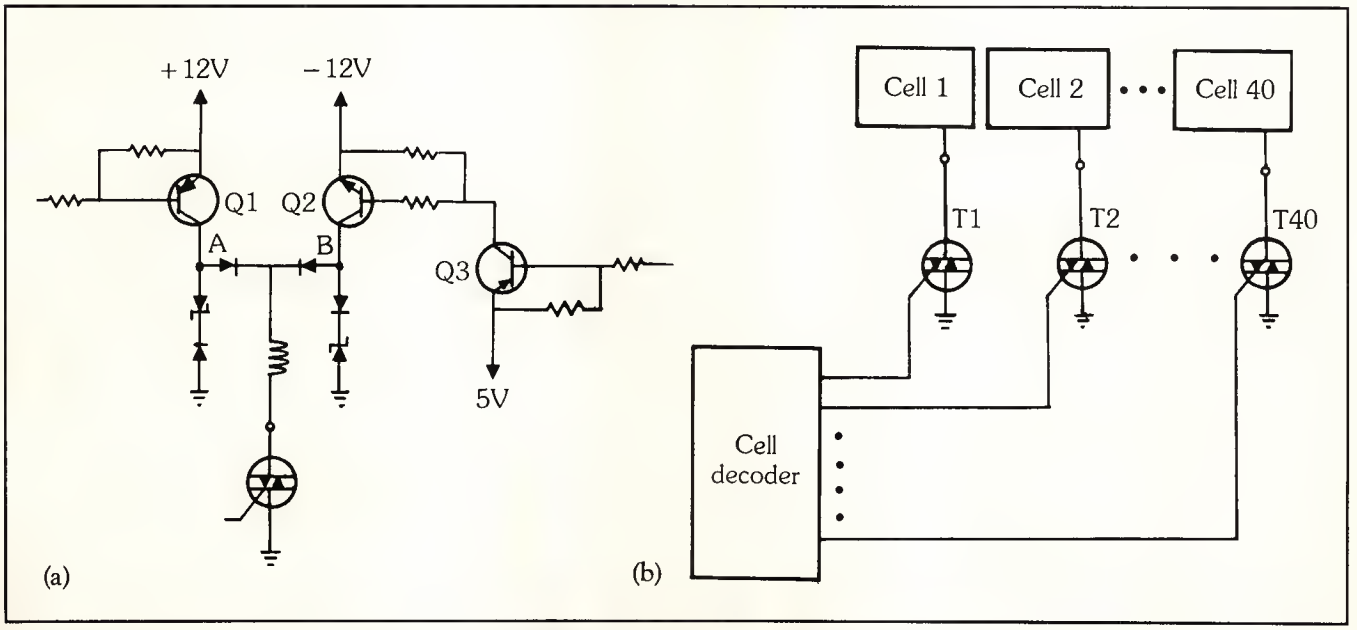


Figure 6. Drive circuit for a braille cell solenoid (a); selection circuit for a braille cell (b).

solenoid. As long as the common point is grounded via a triac, a positive current flows and lowers the pin. Similarly, when Q2 conducts (achieved by turning on Q3), a negative current flows and raises the pin. Figure 6b shows how the common point of a cell is grounded by means of triacs T1, T2, . . . , T*n* and the cell decoder. The circuit between points A and B in Figure 6a is redrawn in Figure 7. Here the two diode pairs D3, D4 and D5, D6 are used to sup-

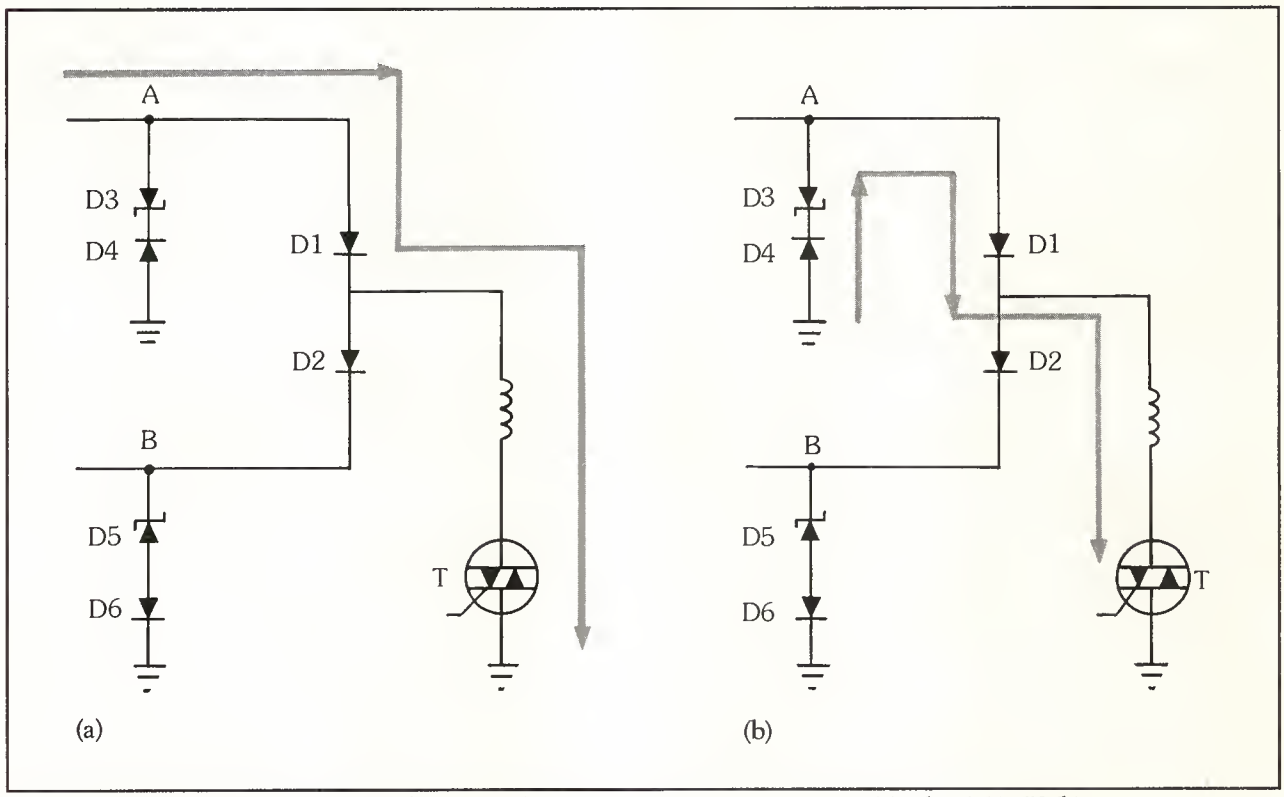


Figure 7. Back-EMF path—normal current flow (a); current flow produced by back-EMF (b).

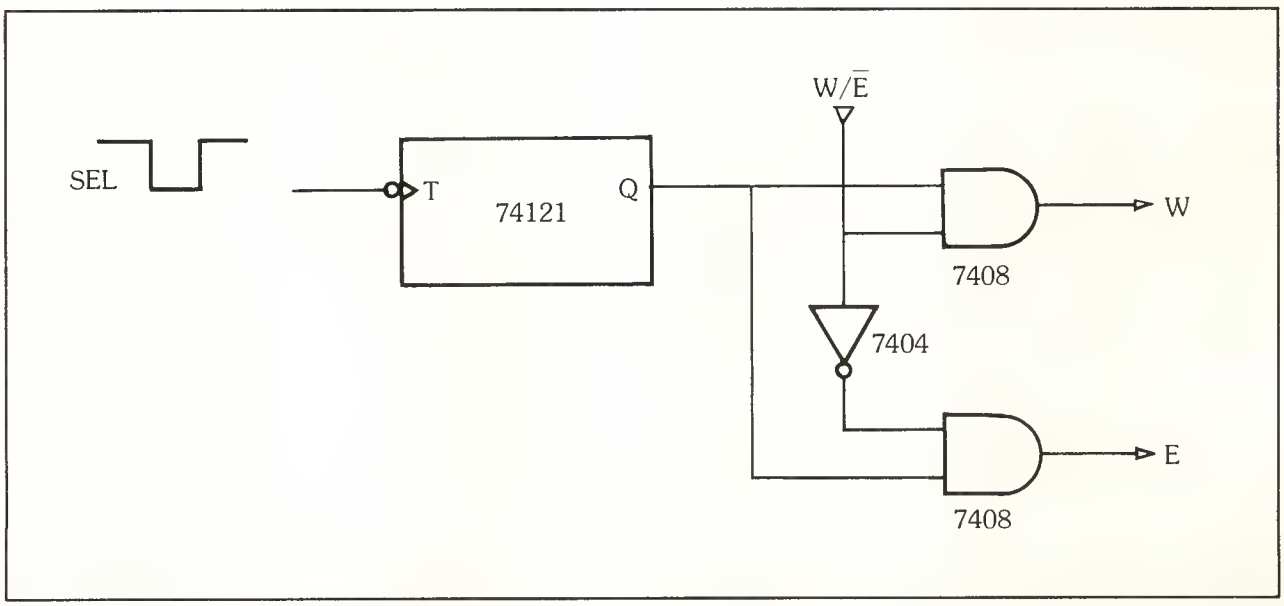


Figure 8. Timing pulse generator.

press the side effects of the back-EMF produced by switching the current through the solenoid on and off. Figure 7a indicates the normal current flow, whereas Figure 7b depicts the current flow produced by back-EMF. Operational details of this circuit can be found in Schena and Grossner. [8]

As can be seen in Table 2, the braille transducer requires a 10-millisecond timing pulse for proper operation. This pulse is generated by the circuit shown in Figure 8, by means of a monostable multivibrator (a 74121) that is triggered by the rising edge of a select (SEL) pulse. The SEL pulse is also used to latch data into the data register. As a result, writing into the data register automatically causes the braille character to be displayed.

Control software

The control software installed on the SBC performs the functions shown in Figure 9. The system executive coordinates the functions of all the modules. Commands sent to the SBC are processed by the command processor. In a modularly expandable system, one must be able to configure the system to meet the specific needs of the current user and to initialize all hardware registers accordingly at the time of power-on and reset operations. Hence, the system configuration module, upon power-on or reset, reads the settings of the function switches (see Figure 3), determines the desired configuration, and initializes the system.

The translator is responsible for the translation of incoming ASCII text into the appropriate braille code. Translation to Grade I braille involves verification of punctuation, insertion of composite signs, and verification of the order in which punctuation and composite-sign indicators occur. Translation into Grade II braille is context-sensitive and requires extensive pattern matching. Hence, this capability is not included in our prototype.

The braille display driver, printer driver, and SBC-to-host communication driver shown in Figure 9 are used by the SBC to communicate with the outside world. Of these three modules, the latter two are initiated through interrupt

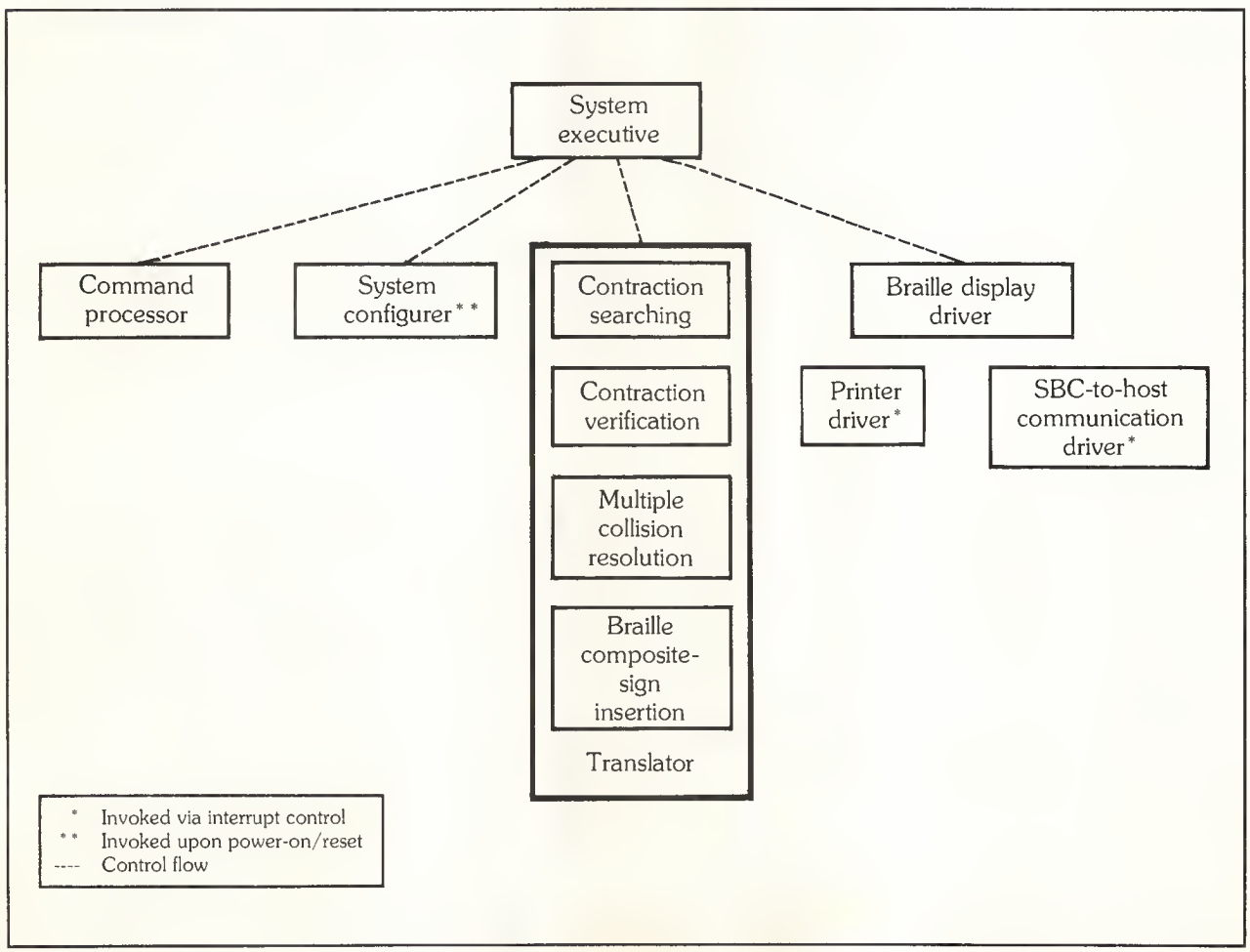


Figure 9. Control software structure.

control. The braille display driver is responsible for taking buffered hex codes representing braille characters and writing them onto the braille display. The translator generates the hex codes by successively loading the cell and data registers and polling the status register.

Development of the control software for the SBC was facilitated by the characteristics of the Z8671. It is a single-chip computer with a Tiny Basic interpreter and debug monitor in its ROM, and it provides 47 assembly language instructions and nine addressing modes. (Complete details of the Z8671 can be found in the Z8 manual.[9]) Using the Basic debug monitor, one can create assembly language subroutines that can be called from a Basic program. In this manner, one can write time-critical portions of the control software in assembly language rather than in Basic, which is interpreted and inherently slow. In addition, there is a cross-assembler for the Z8671 on CP/M-based machines. It allows a programmer to use more sophisticated software tools for the development of the control software. Using the Basic monitor, we developed a bootstrap loader which we used to download control programs prepared on a CP/M computer.

The production cost of a display unit consisting of 40 cells should be about $2000, when the unit is produced in small quantities. For only about $1200, one should be able to build a 20-cell display device.

Application layer

To make full use of the braille display described here, a personal computer must have certain types of software support. In a personal computer, the console or CRT is used to display four types of output:

(a) echoed user-typed information (commands, program statements, and data),
(b) operating system prompts and sign-on messages,
(c) diagnostic and other reports from software tools such as compilers, editors, and debuggers, and
(d) program outputs.

It should be noted that a conventional CRT displays 24 lines of 80 characters each and that many operating systems and software tools are written with this display capability in mind. However, a braille display device of the same size as a CRT, that is, 24×80 characters, would be prohibitively expensive. Hence, we have to live with a line-oriented braille display device. With such a device, the display of type b and c output will require changes in the personal computer's operating system and in its console drivers.[1]

For d-type output in braille, a set of library subroutines that can be called by user-written programs can be provided on the host computer. These routines can perform functions such as routing user output to the SBC, commanding the SBC to display a line on the braille display, and halting the display until the user is ready to read the next line.

An option that is desirable in some circumstances is a turnkey or black box system that allows a user to perform a specific task. Such a task could be text-to-braille conversion, word processing with a line-oriented output as an interface to videotex,[10,11] or airline reservation making. A black box system will have to map type a, b, c, and d outputs onto a line-oriented display device. We hope that several black boxes, in the form of application software packages, will be developed for tasks for which the visually handicapped can be gainfully employed, and that such packages will be distributed through an easily accessible clearinghouse.

We have described an intelligent braille display device controlled by a dedicated single-chip microcomputer. We have built a prototype of this device in our laboratory, using an Apple II personal computer as the host, and have developed driver software for it. We have found the operation of our braille display device to be very satisfactory.

More than 80 percent of the cost of our device is in its electromechanical display. The cost of the display is about $32 per braille cell. We believe that a noteworthy and useful feature of our device is that the number of its braille cells can be expanded modularly. This flexibility accrues from the programmability of the SBC. The production cost of a display unit consisting of 40 cells would be about $2000, when the unit is produced in small quantities. For only about $1200, one should be able to build a 20-cell display device.

References

1. C. P. Grossner, T. Radhakrishnan, and A. Pospiech, "An Integrated Workstation for the Visually Handicapped," *IEEE Micro*, Vol. 3, No. 3, June 1983, pp. 8-17.
2. *The Versabraille System Model 2 User's Manual*, Telesensory Systems Inc., Palo Alto, Calif.
3. R. S. Nickerdon and K. N. Stevens, *Approaches to the Study of Relationships Between Intelligibility and Physical Properties of Speech,* tech. report, National Technical Institute for the Deaf, Rochester, N.Y., June 1979, pp. 20-23.
4. *Vortrax Personal Speech System User's Manual*, Vortrax Corporation, Troy, Mich., 1983.
5. *SC-01 Speech Synthesizer Data Sheets*, Vortrax Corporation, Troy, Mich., 1981.
6. M. B. Dorf and E. R. Scharry, *Instruction Manual for Braille Transcribing*, American Printing House for the Blind, 1971.
7. *Modular Braille Transducer Data Sheets*, Tiflotel, Viale De Gasperi, Italy.
8. A. Schena and C. Grossner, *A Programmable Dedicated Controller for a Soft Copy Braille Display*, tech. report, Computer Science Dept., Concordia University, Montreal, Que., 1985.
9. *Z8 Microcomputer Technical Manual*, Zilog Inc., Campbell, Calif., 1983.
10. T. Radhakrishnan and A. Madras, "Voice Based Program Editor for Visually Impaired Persons," *J. Visual Impairment and Blindness*, Nov. 1984, pp. 436-437.
11. J. Gecsei, *The Architecture of Videotex Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1983.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Interest Card.

High 150 Medium 151 Low 152

Clifford P. Grossner is a systems analyst and a member of the part-time faculty in the Department of Computer Science at Concordia University. His areas of interest include microcomputer system design and applications, distributed computing, distributed expert systems, and bit-slice CPU design.

Grossner obtained his BS and MS in computer science from Concordia University in 1980 and 1982. He is a member of the IEEE Computer Society.

Thiruvengadam Radhakrishnan is an associate professor of computer science at Concordia University, Montreal. He obtained his BE, with honors, from Madras University in 1966 and his M.Tech and PhD degrees from the Indian Institute of Technology at Kanpur, India. His interests include microprocessor-based systems, local-area networks, computer applications to the handicapped, and man-machine communications in database applications.

Alex Schena is a software engineer for the Avionics Department at CAE Electronics Ltd. His areas of interest include multimicroprocessor systems and digital circuit design. Schena received a bachelor's degree in computer science from Concordia University in 1985, where along with coauthor Grossner he designed and implemented the display controller described in this article.

Questions about this article can be directed to Radhakrishnan at Concordia University, Department of Computer Science, 1455 de Maisonneuve Boulevard West, Montreal, Quebec H3G 1M8, Canada.

Mathematical Software in Basic: RV, Generation of Uniform and Normal Random Variables

Since the work of von Neumann, Metropolis, and Ulam in Los Alamos during the Manhattan Project, the "monte carlo" method has been an important tool for the simulation of processes in the physical and social sciences. These computer modeling experiments require the availability of numbers chosen from one or more statistical distributions. The generation of *pseudo-random numbers,* as these have been named, is now a highly developed science.

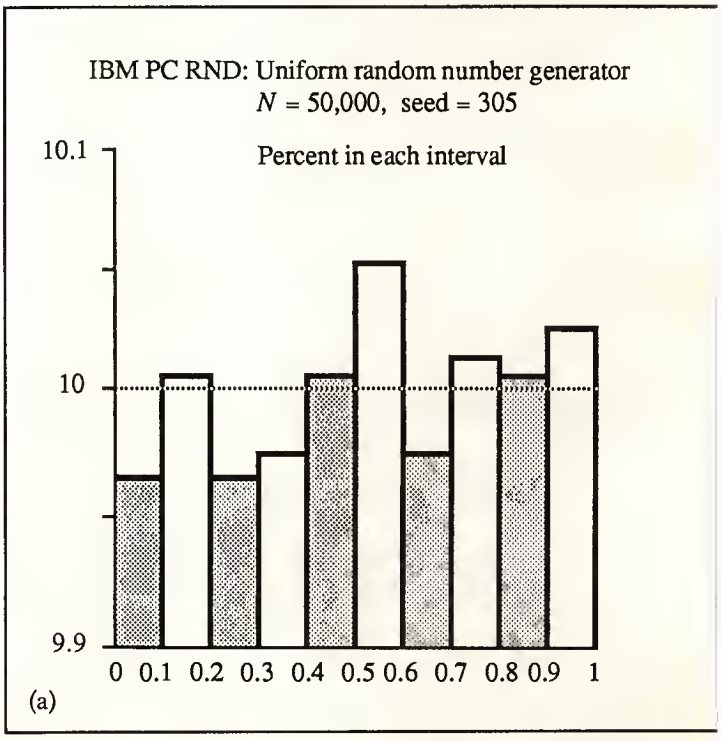
The original concept of obtaining the numbers as output of a physical stochastic process has been replaced by that of algorithmic generators, that is, generators that compute one number after another according to a fixed scheme. Consequently, the numbers are reproducible and thus deterministic. We think of them as equivalent to random numbers if they satisfy various statistical tests of randomness, which include bins, autocorrelations, runs, probability plots, chi-square tests, and so on. However, the collection of such tests is finite and there is always the possibility that some unusual failure lies yet to be discovered. Furthermore, all generators are cyclic, that is, they eventually repeat. A "perfect" generator has the following properties:[1]

- *High quality.* It passes all the tests and has an extremely long period.
- *Efficiency.* Execution is rapid and storage requirements are minimal.
- *Machine independence.* The algorithm works on different kinds of computers; especially, no operation will cause overflow.
- *Flexibility.* The user can specify the number of random bits in the returned number; failing to do that, the generator will give as many bits as possible. Also the user can restart the generator at any time, but explicit initialization is not necessary. A slight change in the starting value will result in a different random sequence.
- *Repeatability.* Specifying the same starting conditions will generate the same sequence.
- *Portability.* The same sequence of random numbers can be produced on different computers or different interpreters/compilers by initializing the generator in exactly the same way.
- *Simplicity.* The algorithm is easy to implement and use.

Here we describe two generators, UNI for uniform random numbers on the interval [0,1], and RNOR for normal (Gaussian) random numbers with zero mean and unit standard deviation, $\mu,\sigma = (0,1)$. They are not "perfect" but do satisfy most of the properties above. The development of these generators is the result of a series of ongoing conversations with George Marsaglia of Washington State University. Marsaglia, who has been at the forefront of research in this area for more than two decades, decried the lack of very high quality generators for the microcomputer user. We have made free and frequent use of his ideas for UNI and for RNOR.[1,2]

Our contribution mostly amounts to a careful implementation and testing. In the following two sections we outline the algorithms. We deliberately omitted many details for reasons of space. A few of these can be found in the two references above. Readers who are interested mainly in the use of the generators can skip the remainder of this article, save for the remarks below about the contents of the disk that is available.

Both UNI and RNOR have been developed in the Basic programming language to the same high standards as two



Two algorithmic generators available on PC-compatible disk have successfully been moved to other computers.

David K. Kahaner, Jeffrey Horlick, Debra K. Foer
National Bureau of Standards

other pieces of mathematical software.[3,4] Basic was chosen because it provides access to these algorithms by low-end users who heretofore have had to rely on either vendor-generated ROM routines or nonportable methods written in assembly language.

Response to our other Basic software has been overwhelmingly from users of the IBM PC family of microcomputers. As a result we have prepared a PC-compatible disk, which we call RV (random variable), containing a demonstration program for the package. The disk also contains a portable Basic source for the generators so that they can be moved to other computers. We have moved them to both a Tektronix 4054 and a Macintosh and subsequently verified that they run well.

The disk also contains programs that will allow a user to generate a sequence of uniform or normal numbers and write them onto a disk file for later use. Several files on this disk are in three forms, a Basic "source" file, which can be loaded and run with the Basic interpreter; a compiled version, which will run seven to eight times faster; and another compiled version, which utilizes the 8087 math coprocessor on the PC if this is available. The files are carefully labeled.

Both generators are modifications of generators that we have written in the Fortran language for use in our public domain mathematical software library for larger computers, CMLIB. Scientists interested in the Fortran or Basic versions should contact the first or second author.

UNI, a uniform generator

For writing simple game programs or some scientific applications where a good sequence of random numbers is not needed, we have found that the built-in random number generator, which is available with most Basics, is both adequate and very fast. An example is the RND function that is available in IBM BasicA for the IBM PC family. In Kahaner and Wyman[3] we used the built-in random number generator. For that application only a very few, at most a dozen, numbers are needed, and the vendor-supplied generator is acceptable. Usually written in assembly language or in ROM, it is never portable.

Such generators often fail some tests for randomness. One simple test is to generate a long sequence and to place each number in a "bin." At the conclusion of the generation we can count the fraction of numbers in each bin. A χ^2 test can be used to see if these fractions deviate much from the expected number. For very bad generators, even a visual inspection of the distribution of numbers in each bin will show improbable deviations. Luckily, all the built-in generators seem to pass this test without difficulty.

Figure 1a illustrates the output of 50,000 calls to RND from the BasicA interpreter, grouped into 10 bins. The between-bin variation is seen to be small and well within statistical variability. A slightly more sophisticated test in-

(b)

Figure 1. 50,000 calls to IBM BasicaA RND (from demo disk), interpreter version. Histogram of frequency of points within one of 10 bins (a). Points plotted as pairs in the unit square with a very high correlation between adjacent points (b).

volves considering the generated numbers in pairs. If we call the generator's outputs $x_1, x_2, \ldots$, we know that

$$0 \leq x_i \leq 1.$$

Usually, neither zero nor one is a possible output, but this is not significant. The pairs $(x_i, x_{i+1}), i = 1,3,5,\ldots$ can be thought of as points in the unit square. Uniform numbers will eventually fill the square smoothly without bunching. Figure 1b illustrates the output of 50,000 calls (25,000 points) to RND from the BasicA interpreter. As is easily seen, there is a very clear pattern of bunching. Each generated number is strongly correlated to the previous one. A simulation to compute a two-dimensional integral using RND in this fashion would miss significant detail in the integrand. Interestingly, the deficiencies with RND only manifest themselves in the interpreter (IBM BasicA, Version 2.1). The compiler (IBM Basic Compiler, Version 1.00) uses another generator, producing different numbers. This in itself violates the portability property mentioned above. Programs developed using RND will execute very differently between the interpreter and compiler.

We will not trace the development or theory of uniform generators here; this is available in standard texts and survey papers.[5-7] Proper application of the theory allows very good generators to be produced. However, Marsaglia has shown that even these generators will eventually show "nonrandom" behavior when subjected to progressively more stringent tests. The implementation of RND in IBM BasicA fails most tests.

Since uniform generators are very widely described,[6] we give only a brief description of ours.

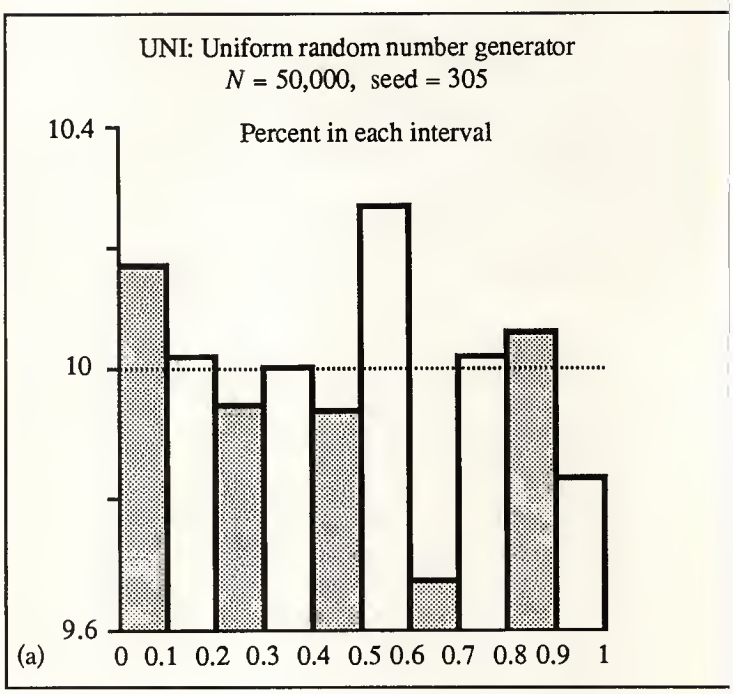
UNI begins with a 17-entry real array whose elements, $x_1, \ldots, x_{17}$, are computed from an initializing integer "seed." The next value (which is also the output value) is obtained from x_{17} and x_5 by subtraction, although any other binary operation would work equally well. (We chose the array length 17 and the combining index 5 based on experience and extensive testing. We know of no theoretical work to suggest that other combinations might not work equally well.)

If the result of the subtraction is negative, 1.0 is added to assure an output on the interval (0,1). The new number is placed at the end of the list, and the first is dropped so that only 17 remain. (This is done with index pointers.) Generators of this type, which combine two past values to obtain the next, are called "Fibonacci." The seed value is used to generate the initial 17 entries, so that each of these has 23 random mantissa bits. These 23·17 bits are set either 0 or 1, depending on whether the integer variable T is positive or not. The T's are generated from the seed as a sequence of random integers in the range (−32767,32767) by the following scheme:

C = *seed*, A = 1947, B = 1111, M = 32707
Initialize all mantissa bits to 0.

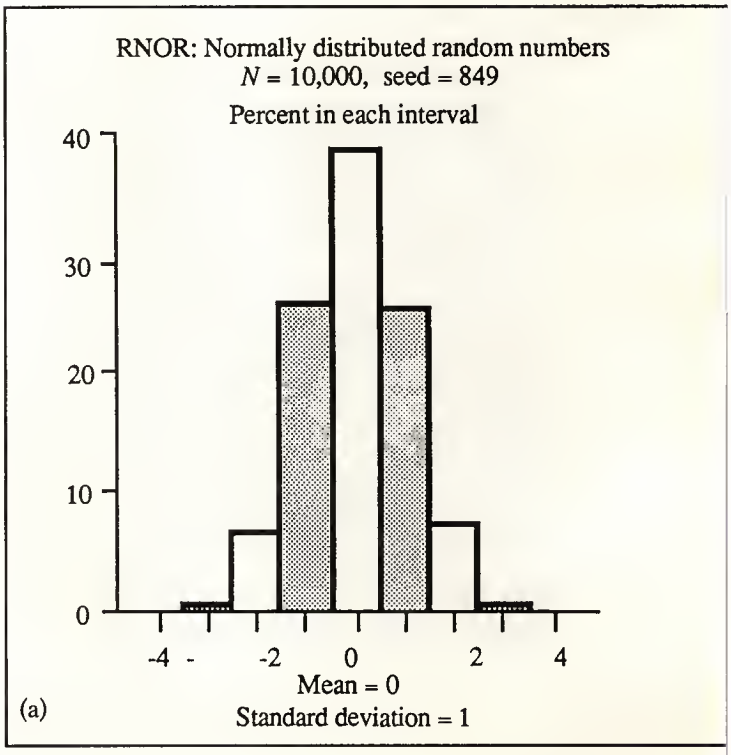
Repeat until all bits are set:

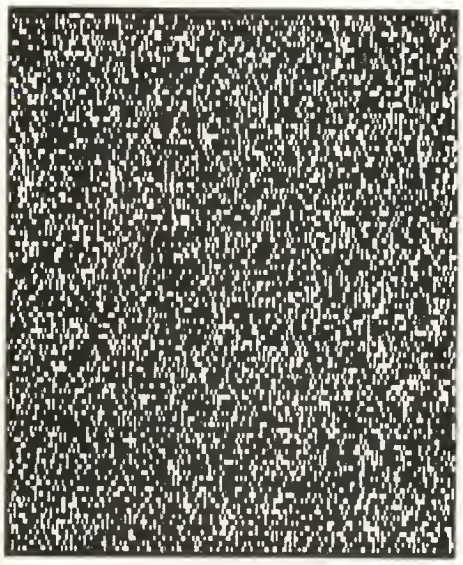
$$T = A - C$$



if $T<0$, set next bit to 1, else set to 0
IF $T<0$, THEN $T = T+M$
$C = B, B = A, A = T$

The bit setting is done with floating-point multiplications by 0.5, which are exact. The other arithmetic uses integers that can be represented exactly using eight bits.





(b)

Figure 2. 50,000 calls to UNI (from demo disk), interpreter or compiled version. Histogram of frequency of points within one of 10 bins (a). Points plotted as pairs in the unit square with an apparent lack of correlation between adjacent points (b).

The results ought to be completely reproducible on any computer supporting at least 4-byte floating-point numbers with a 23-bit mantissa and eight-bit integers. Hence UNI is very portable.

UNI also has a very long period before it cycles. The period can be shown to be

$$2^{n-1}\left(2^{17}-1\right),$$

where n is the number of bits in the fractional part of x_i that is, the signed mantissa. For the IBM PC family $n = 24$ and thus the period is more than one *trillion*. UNI is fairly fast—fast enough to use in place of RND in many applications—and gives far better results. There are faster algorithms, but they are generally less portable and may be less robust.[8] Figure 2 (from the IBM demonstration disk) illustrates the output of 50,000 calls (25,000 points) to UNI. Comparison with Figure 1a will indicate that UNI has somewhat more variation in the bins than RND (although this is well within statistically allowed fluctuations as measured by the χ^2 test) but no noticeable correlation. This figure by itself cannot justify our confidence in UNI. But we have subjected it (successfully) to all the practical stringent tests we know.

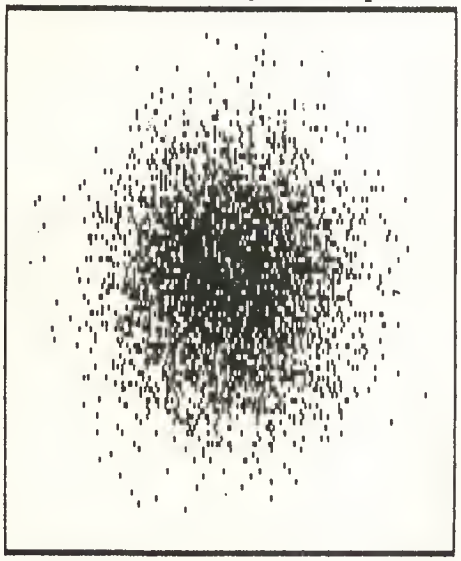
RNOR, normal random number generator

A variable X is said to be normally distributed with mean μ and standard deviation σ if its distribution function is given by

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{x} e^{-(t-\mu)^2/2\sigma^2}dt.$$

We denote this $N(\mu,\sigma^2)$. The graph of this distribution function rises smoothly from zero to one as x increases from $-\infty$ to $+\infty$. The graph of the density of this variable (the actual integrand above) is the familiar "bell-shaped" curve.

To our knowledge all normal generators start with values from a uniform generator.[5] Thus the quality of the resulting normals is directly related to the quality of the uniforms. RNOR, which produces normally distributed random numbers with mean zero and unit standard deviation, has UNI built in. The same demonstration program that shows UNI also illustrates RNOR. Pairs of numbers (x_i,x_{i+1}), $i = 1,3,5,\ldots$ are plotted in a square centered at the origin. This is presented in Figure 3 (from the same demonstration program which produced Figures 1 and 2)



(b)

Figure 3. 50,000 calls to RNOR (from demo disk), interpreter or compiled version. Histogram of frequency of points within one of 10 bins . Histogram should appear approximately normal (a). Points plotted as pairs in the unit square with the traditional buckshot pattern of the bivariate normal (b).

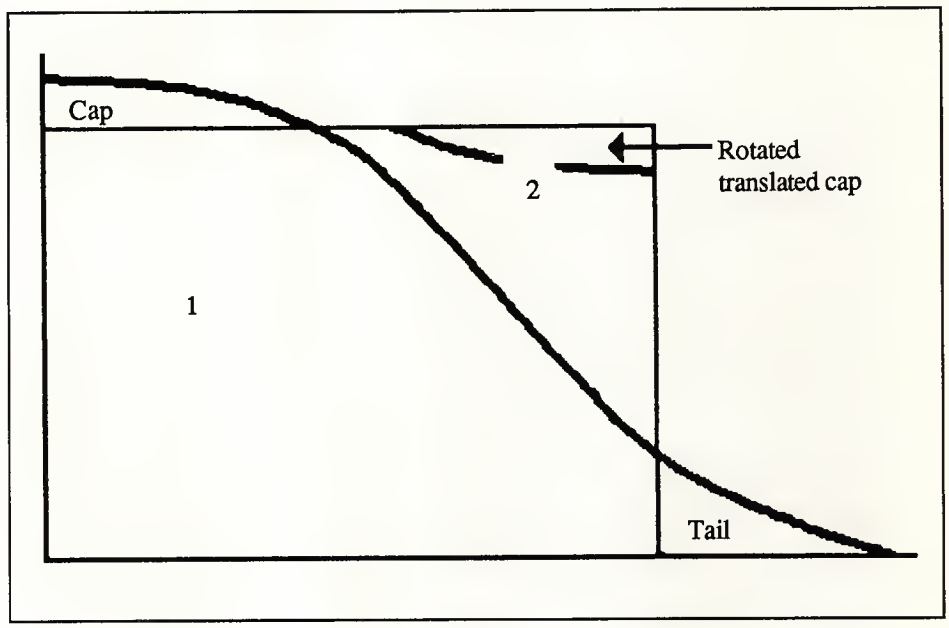


Figure 4. A density composed of four regions. The area under the density equals the area of the rectangle. To sample from the density, we can sample in the rectangle but must determine in which region the point lies.

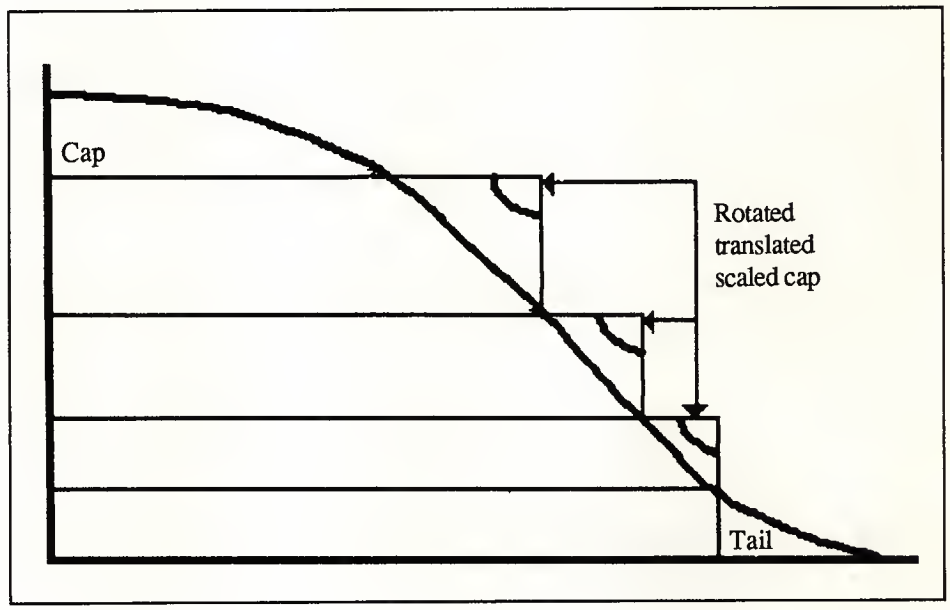


Figure 5. A ziggurat with four layers. To sample from $n(x)$, pick a slice and (horizontally) a point X in the slice. If the latter lies to the left of the position $n(x)$ and enters the slice, return X. If not, pick Y in the slice. If (X,Y) is under $n(x)$, return X. If it is in the scaled, rotated cap, return $b[x(j) - X]$. Otherwise, it is in the tail and we must sample from that directly. This is done quickly if the area in the tail and scaled cap is small.

and illustrates the familiar "buckshot" pattern of the bivariate normal with correlation coefficient zero.

There are two well-known methods for generating normals. One involves computing the average, $\bar{x}_p$, of p uniforms, each generated on the interval $[-1,1]$. If p is large, the *Central Limit Theorem* states that $\bar{x}_p\sqrt{3p}$ is approximately normal with mean 0 and variance 1. Historically, p is taken as 12. This process is expensive in that 12 uniforms are required for each normal. The resulting density function also has slightly smaller tails than the normal; it is not recommended. The second method, due to Box and Muller (in Section 6.5)[9] begins with U and V uniform on [0,1], and forms $S = \sqrt{-2 \ln U}$. Then the numbers $X = S\cos(2\pi V)$ and $Y = S\sin(2\pi V)$ are both $N(0,1)$.

A variation due to Marsaglia (again Section 6.5)[9] and later Bell[5] is to generate two uniforms on $[-1,1]$, say U and V, and compute $R = U^2 + V^2$. If $R > 1$, discard U and V and regenerate a pair of uniforms. (We lose 22 percent of our efficiency here.)

If $R \leq 1$, compute $W = \sqrt{-2 \ln R/R}$. Then both $X = U \cdot W$ and $Y = V \cdot W$ are independent and $N(0,1)$. While this method is only 78 percent efficient, it avoids the calculation of a sin and cos. Because of its simplicity,

we recommend this procedure for those cases where time is not a serious issue.

There is a third, more general, method.[9] Generate Y uniformly between 0 and 1. Find X, the upper limit of the integral for $G(X)$, so that $G(X) = Y$. Then X is normally distributed $N(\mu,\sigma)$. The only hitch is computing X, which is exactly equivalent to finding the "inverse" of the normal. This cannot be done in closed form. Some kind of further approximation is required. One way to generate the approximation is to replace one distribution by another that is composed of segments of straight lines or other simple curves. The disadvantage is that values generated by this new distribution are only approximately normal.

The remainder of this section is devoted to a detailed description of the method we use in RNOR. Understanding this material is not prerequisite to effective use of the program, and some readers may wish to skip to the following section.

In this description we consider only the right half of the normal and rescale so that this half has unit area. Call $n(x)$ this density. To generate a variable with normal density, first pick a point t uniformly on the y axis between 0 and $n(0)$. Then at height t pick another point uniformly between 0 and the intersection with $n(x)$. The latter point has the requisite distribution, but the direct method for finding it requires too much computation to be useful.

Construct a rectangle R of unit area with sides on the positive x and y axes. The upper right corner $P = (a,b)$ of R is not yet specified. We can pick this corner so that $n(x)$ passes into and out of R. This leads to four subregions (see Figure 4):

1. The region inside R and below the curve $n(x)$.
2. The region inside R but above $n(x)$.

Cap. The region above R and below the curve $n(x)$. This begins at $x = 0$.

Tail. The region to the right of R and below the curve. This extends to ∞.

By adjusting P we can make the cap as small as we like. The cap can be removed and slid around without lifting it off the plane so that it fits in the upper right corner of region 2. The remainder of region 2 will then have the same area as that of the tail.

To generate a variable with the same density as n, that is, normal, pick a point $T = (X, Y)$ uniformly in R. This can be done by first choosing a number uniformly between 0 and b to give Y, and then another number uniformly between 0 and a to give X. If T is in region 1, return X. If T is in the rotated cap, return $a-X$. T will not be in those with probability equal to the area of the tail. In that case we must generate a new number from the density associated with the tail.

Excluding the tail, this technique will be efficient only if it's easy to decide if T is in region 1 or, if not, if it's in the rotated cap. For some values of T this is very simple. For example, if T is less than the point where $n(x)$ intersects R, it is obviously in region 1. The essence of the *ziggurat* method[2] is that, by cutting up $n(x)$ judiciously, we can make such decisions very simply most of the time. Those cases when it is difficult (and time-consuming) to decide occur infrequently.

The concept of the ziggurat comes from what we have to do to make this technique practical. The ziggurat function is a "down-staircase" function composed of k layers, each with area $1/k$. Figure 5 shows a ziggurat with four layers. To pick a point randomly under the ziggurat, select j from $1,2,\ldots,k$ with a uniform generator and then pick a point X in the jth layer.

RNOR uses a ziggurat to approximate the normal curve $n(x)$ leading to a fast generator. To do this, we superimpose a ziggurat on the density we are really interested in sampling so that each instep of the stair intersects the density $n(x)$. The following describes how the 64-layer ziggurat was chosen for RNOR. The steps of the ziggurat, labeled from left to right $x_1,x_2,\ldots,x_k$ are computed in advance. We work our way up from right to left (see Figure 5). The kth, or bottom, layer extends from $x = 0$ to $x = x_k$ and has area $1/k$. Thus we have

$$\frac{1}{k} = n(x_k)x_k.$$

Because we want the density n to be under the ziggurat, we also pick $x_{k-1} = x_k$. Then x_{k-2} satisfies

$$\frac{1}{k} = \left[n(x_{k-2}) - n(x_{k-1})\right]x_{k-1}, \text{ or}$$

$$n(x_{k-2}) = n(x_{k-1}) + \frac{1}{kx_{k-1}}.$$

In general

$$n(x_p) = n(x_{p+1}) + \frac{1}{kx_{p+1}}, \quad p = k-1, k-2, \ldots, 1.$$

This tells us where the steps on the ziggurat have to be. Computing these numbers is relatively time-consuming, so it is done only once and the values are stored in the program. This construction leaves a cap above the first layer. The area of the cap is, thus far, excluded from the ziggurat. We include it by placing shrunken copies of the cap into each layer in a proportional way. The proportionality factor is computed in Equations 1 through 4 below.

In each layer we construct a small rectangle R_j at its right end, which has upper left and lower right corners

$$[x_{j-1}, n(x_{j-1})] \quad [x_j, n(x_j)]$$

respectively, and then translate and expand R_j to the unit square. Next, rotate and translate the cap to the upper right corner of this square. In addition, shrink this image of the cap by an amount b in x and $b[n(0)—n(x_0)]/x_0$ in y. The curve that then appears in the upper right corner of the square has equation

$$1 - \frac{x_0}{b} \cdot \frac{n(b-bx) - n(x_0)}{n(0) - n(x_0)}. \qquad \textbf{(1)}$$

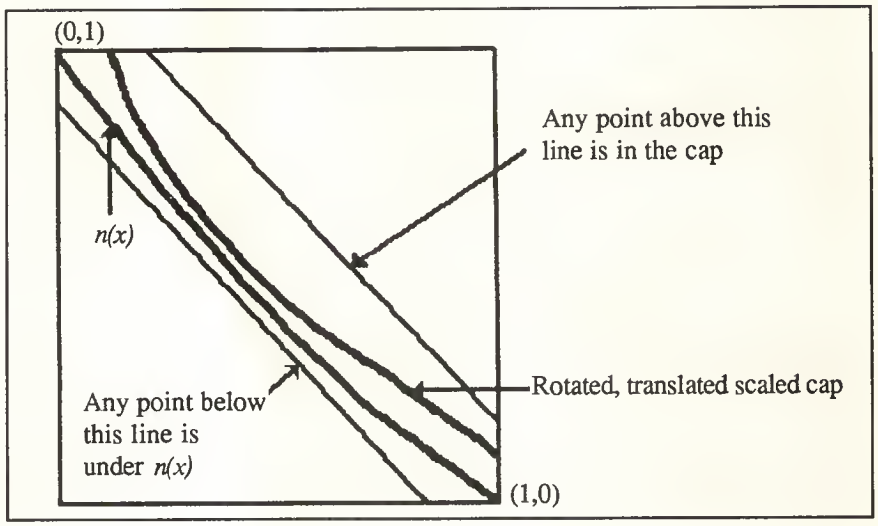


Figure 6. Details of mapping a layer into the unit square.

By using enough layers, that is, choosing k large enough, we can be assured that the scaled cap lies entirely above the diagonal of the square. With $k = 64$, $x_k = 2.78$, the areas of the cap and tail are 0.133 and 0.003 respectively. The region in the unit square above this curve has area

$$r = \frac{x_0}{b^2[n(0) - n(x_0)]} \cdot (\text{cap area}). \tag{2}$$

Finally, move the square back onto R_j.

How shall we select b? The criterion is that the total area in the R_j's occupied by the the scaled caps should equal the total cap area. The area of R_j is

$$(x_j - x_{j-1}) \cdot [n(x_{j-1}) - n(x_j)]$$

$$= (x_j - x_{j-1}) / kx_j = \left(1 - \frac{x_{j-1}}{x_j}\right) / k,$$

the area of the scaled cap will be r times this,

$$\frac{r}{k}\left(1 - \frac{x_{j-1}}{x_j}\right),$$

and the sum of these is required to be the cap area,

$$\frac{r}{k}\left[\left(1 - \frac{x_0}{x_1}\right) + \ldots + \left(1 - \frac{x_{k-1}}{x_k}\right)\right]$$

$$= \text{cap area}. \tag{3}$$

Equations 2 and 3 can be combined to eliminate r and the cap area and to tell us what b must be,

$$b^2 = x_0\left[\frac{1 - (1/k)\Sigma_{i=1}^{k} x_{i-1}/x_i}{n(0) - n(x_0)}\right] \tag{4}$$

for $k = 64$, $b = 0.488$.

Now that the ziggurat has been constructed, we will demonstrate its use. Pick a point (X, Y) uniformly in the ziggurat. If it lies under $n(x)$, return X. If it lies inside one of the scaled caps, return the abscissa value corresponding to it in the original cap. The only remaining alternative is that (X, Y) is above $n(x)$ but below the cap, and this occurs with probability equal to that of the tail. Thus we must return an X from the tail. The procedure for doing this is described below. Combining these three cases returns a set of X's having the same distribution as n the normal. An equivalent, but more algebraic, description of the method is

1. Pick j uniformly from 1,2,...,k.
2. Pick an X uniformly between 0 and x_j. If $X < x_{j-1}$ (as it will be most of the time), return X.
3. Otherwise pick Y in the jth layer, that is, between $n(x_j)$ and $n(x_j) + 1/kx_j$. If Y is under $n(x)$, return X. If Y is in the rotated, scaled cap, return $b \cdot (x_j - X)$. If Y is not in any of these regions, it has the same probability of occurrence as the area of the tail, and we must sample from that directly.

We use a ziggurat with 64 layers. There is enough information in one call to UNI to pick a layer and also an X inside the layer. Recall that UNI operates mostly with integers and "$I + I$ mod 64" will generate a layer. With 64 layers about 96 percent of the calls to RNOR result in $X < x_{j-1}$. *Thus in 19 out of 20 calls only one uniform is required to generate a normal, and in these cases no special functions such as sin, ln, or even square root are required.* Using 64 layers also means that inside R_j the density n and the image of the cap are almost linear functions. It's possible to approximate these images so that evaluating $n(x)$ is not necessary for most of the remaining 4 percent. That is, if the Y selected in step 3 is above or below an appropriate line (see Figure 6), it is not necessary to evaluate $n(x)$ to verify the region in which it

lies. This is a speedup procedure; it has no effect on the results.

If (X, Y) is in the portion of R_j corresponding to the tail, we have to sample from a density equal to that of the tail scaled up so its total area is 1. This can take some extra work and time, but we need not be too concerned because we fall into this region very infrequently. Sampling from the tail is done by a "rejection region" technique.[9]

We use the function $g(t) = \exp(-x_k t)$, scaled so it lies just slightly above the tail. We can sample from this kind of distribution by using the inverse function approach described at the beginning of this section. That is, given a uniform Y, we can find (analytically) an X such that

$$\int^{x} \exp(-x_k t) = Y.$$

This X is an acceptable one to return as long as a uniform multiple of Y lies under $n(X)$. If this fails, we have to pick another Y and try again. Thus sampling from the tail distribution requires generation of *at least* two uniforms and evaluation of the target density n.

Generation of uniform numbers is required in the tail and may be needed in the cap. The most portable way to do this is to use UNI. Since this occurs very infrequently, we have elected to use the built-in uniform generator (RND in BasicA) for these calls. We have not detected any statistical degradation caused by using RND here. We do not use RND in the selection of the layer or point within the layer because its properties are simply not good enough for this phase of the computation. Users who wish to have the most portable generator may decide to use UNI for all the uniform calls. On the other hand, users who have adequate uniform generators on their computers can reverse the process and replace all calls to UNI by those to a built-in generator to speed up the computation.

Some timings

In discussing random number generators, we have mentioned those characteristics of the "perfect" generator. It will be up to users to determine to what extent UNI and RNOR approach these general criteria. With respect to the issues of speed, we remark that UNI has been designed with an eye to good statistical properties foremost.

Its two most attractive features are high quality and portability. It will run more slowly that most built-in generators but is acceptably fast compared to non-built-ins.

RNOR is a more complicated situation. In analyzing it, we must consider issues of quality, portability, reproducibility, and speed. For the last, the real question is "compared to what?" Since any normal generator requires an underlying uniform generator, the uniforms ought to be identical before any comparisons are made. Generators like Box-Muller require computation of various mathematical functions, such as ln, sin, cos, square root, etc., whose speed differs from one machine to another. These functions are usually "hard wired," and their timing doesn't change much between interpreted and compiled programs. Other parts of a program will change when Basic is compiled.

Table 1.
Sample timings for 10,000 $N(0,1)$ evaluations
IBM XT BasicA and Basic Compiler
(seconds)

	RNOR	B-M-UNI	B-M-RND (GOSUB)	B-M-RND (in-line)
Interpreted	620	637	394	374
Compiled	56	90	74	72

The point of all this is that readers should be very careful when comparing "apples and oranges." We give a set of timings—for illustration only—other combinations of code may give very different results.

A major virtue of the Box-Muller algorithm is simplicity. We wrote a short BasicA Box-Muller subroutine to compute a pair of $N(0,1)$ numbers using UNI as the uniform generator. We also wrote a BasicA main program that calls this 5000 times, generating a total of 10,000 normals. This is compared to a similar main program that calls RNOR 10,000 times.

In both cases we initialized the generators outside the major loop so the initializing time is excluded. The runtimes are given (in seconds) in the first two columns of Table 1 for both compiled and interpreted BasicA. The two remaining columns give times for variations on the Box-Muller implementation. In column 3 we replace UNI with the built-in generator RND while leaving the subroutine call intact. Column 4 replaces UNI with RND and also replaces the GOSUB with an in-line calculation.

We conclude that in compiled Basic one always ought to use RNOR rather than Box-Muller. For interpreted Basic the choice depends on the quality of the built-in uniform generator.

References

1. G. Marsaglia, "Comments on the Perfect Random Number Generator," Notes: Washington State University, Computer Science Dept., Pullman, WA 99164, Nov. 1981.
2. G. Marsaglia and W. Tsang, "A Fast, Easily Implemented Method for Sampling From Decreasing or Symmetric Unimodal Density Functions," *SIAM J. Sci. Stat. Comput.*, Vol. 5, No. 2, June 1984, pp. 349-359.
3. D. Kahaner and W. Wyman, "Mathematical Software in Basic—GLAQ: Evaluation of Definite Integrals," *IEEE Micro,* Vol. 3, No. 5, Oct. 1983, pp. 42-46.
4. D. Kahaner, J. Horlick, and W. Wyman, "Mathematical Software in Basic—Dint: Data Integration," *IEEE Micro,* Vol. 5, No. 2, Apr. 1985, pp. 76-82.
5. D. Knuth, *The Art of Computer Programming: Semi-numerical Algorithms,* 2nd ed., Chap. 3, Addison-Wesley, Reading, Mass., 1981.
6. G. Forsythe, M. Malcolm, and C. Moler, *Computer Methods for Mathematical Computations,* Prentice-Hall, New York, 1977, pp. 240-249.
7. G. Marsaglia, "Random Numbers Fall Mainly in the Plane," *Proc. Nat'l. Acad. Sci.,* Vol. 61, 1968, pp. 25-28.
8. S. Kirkpatrick and E. Stoll, "A Very Fast Shift-Register Sequence Random Number Generator," *J. Comp. Phys.,* Vol. 40, 1981, pp. 517-526.
9. W. Kennedy, Jr., and J. Gentle, Statistical Computing, "Statistics: Textbooks and Monographs," Vol. 33, Marcel Dekker, 1980.

David K. Kahaner is the technical group leader for scientific software in the Scientific Computing Division, Center for Applied Mathematics at the National Bureau of Standards. His research interests include evaluation of integrals, solution of ordinary differential equations, interpolation, Fourier transforms, and mathematical software in general.

Kahaner received his PhD in applied mathematics from the Stevens Institute of Technology, Hoboken, New Jersey, in 1968. He spent most of the next 11 years at Los Alamos National Laboratory as a numerical analyst. He has been a visiting professor in the Mathematics Department of the University of Michigan, at the ETH-Zurich, at the Technical University-Vienna, and at the Catholic University of America.

Jeffrey Horlick received his BS in physics from the University of Maryland. His graduate research was conducted at the University of Manchester, England. Horlick has been employed by NBS since 1967, working in the areas of metallurgy, paper physics, and consumer products. For ten years he worked with the NBS Collaborative Reference Programs, which provide sample materials and statistical analyses for testing laboratory self-evaluation. For the last five years, he has been a project leader in the National Voluntary Laboratory Accreditation Program with responsibility for proficiency testing and data analysis.

Debra K. Foer is an applications programmer for the IBM Information Systems Group located in Bethesda, Maryland. While attending American University in Washington, D.C., she assisted in the design, development, and maintenance of software packages in Basic at NBS's Scientific Computing Division.

Foer received a BS in computer science and a BA in psychology in 1985 from American University.

Questions about this article can be directed to Kahaner, Division 713, Room A151, Building 225, National Bureau of Standards, Gaithersburg, MD 20899.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Interest Card.

High 168 Medium 169 Low 170

A Cordic Processor for Laser Trimming

T. W. Curtis and Paul Allison Pacific Missile Test Center

James A. Howard University of California, Santa Barbara

A coordinate rotation algorithm can be used to correct the position of a microcircuit on a laser trimming platform. A software implementation of the algorithm was too slow, however. Special-purpose hardware solved the problem.

A moderately priced, microcomputer-based system has been used for the last seven years at the Microelectronics Laboratory of the Pacific Missile Test Center, Point Mugu, California, for the laser trimming of hybrid microcircuit resistors.[1] These resistors are formed by the deposition of patterns of resistive material onto a circuit substrate; each is then trimmed to the correct value by a computer-controlled laser. The laser's positioning system allows *x-y* coordinate translation, although any rotation adjustment must be done with software routines. This approach to rotation requires an excessive amount of computation time—on the order of hundreds of milliseconds. A hardware approach is an obvious solution to this problem.

Here, we describe the design and implementation of a custom processor that provides rotational correction in laser trimming. The processor accepts a rotation angle and an *x-y* coordinate pair from the laser-control computer and returns the rotated coordinates. The rotation technique used is a special adaptation of the Coordinate Rotation Digital Computer, or Cordic, algorithm. The computation speed of the processor is comparable to the control computer's I/O instruction time.

The laser trimming system

A functional block diagram of our original laser trimming system is given in Figure 1. The controlling computer consists of three circuit cards interconnected on an IEEE 696

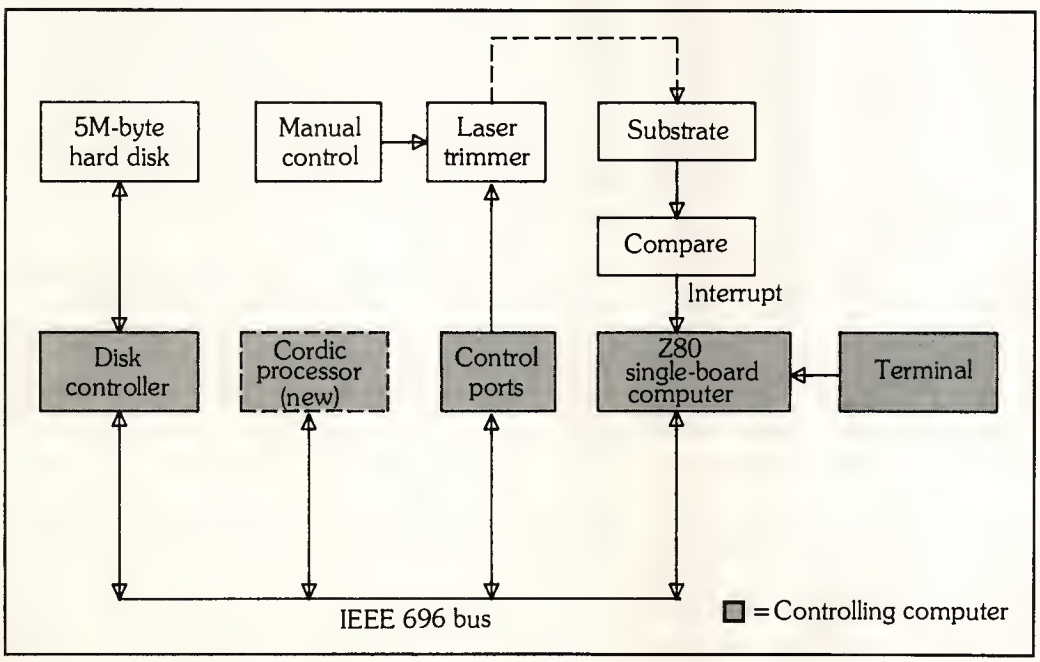


Figure 1. Block diagram of the microcomputer-based laser trimming system.

0272-1732/86/0600-0061$01.00 © 1986 IEEE

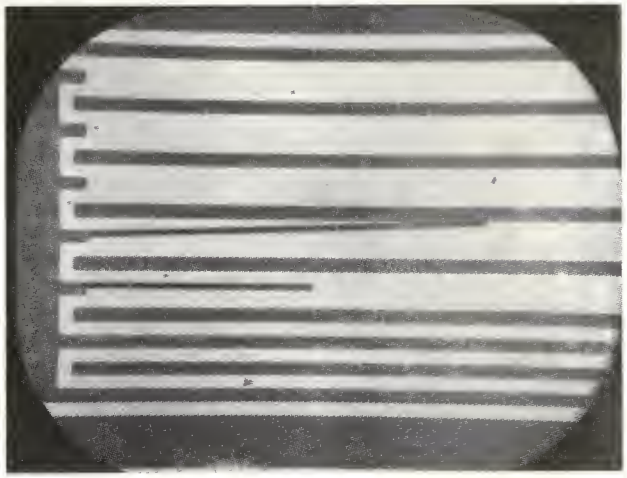


Figure 2. Laser cut or kerf. The top laser kerf shows what happens when a resistor pattern that has been rotated out of alignment is trimmed. The lower kerf represents a correct trim.

bus. A Z80-based single-board computer contains the CPU, memory, and terminal interface. A control-port board supplies the laser trimmer interface, and a hard disk controller connects the mass program storage of 5M bytes. (The Cordic processor that we added to the original system is the card shown with dashed margins in the figure; it becomes a part of the controlling computer and is addressed as a group of I/O ports.)

A circuit substrate is mounted on an *x-y* positioning table beneath the laser beam. The substrate is manually aligned, and the computer then locates each resistor on it according to a preloaded lookup table. The value of each resistor is compared to a reference resistor. The trimming process is interrupted when the trimmed value matches the reference.

The placement of each resistor is relative to two reference marks on the substrate. For proper trimming, the axis of the circuit pattern must be aligned with the axis of the positioning system. Unfortunately, the pattern has usually been rotated out of proper position during mounting, pattern screening, and so on. If the position is not corrected, a long laser cut (termed a kerf) will wander off the edge of the resistor, as shown in Figure 2. (A normal laser kerf is shown in the lower portion of the figure.) Furthermore, the trimming system may not be able to find resistors far from the first reference mark at all if no rotational correction is provided.

Alignment is a semimanual procedure. The operator slews the *x-y* table to the location of the first reference mark and enters this location as 0,0. The computer then slews to the location of the second mark. The operator then corrects the alignment of the circuit pattern and the computer calculates the error angle. The Cordic processor uses this error angle to correct each coordinate pair as trimming proceeds.

A typical substrate with 30 resistors requires up to five cuts and two rotations per cut for each resistor. This equates to a total of 300 coordinate pairs to be rotated to their true positions for each substrate. The software rotation time of the original system is 845 ms per rotation; consequently, a total time of 253.5 seconds, or 4.22 minutes, is required for software-based rotation correction of a substrate. This excessive time makes a hardware Cordic processor very attractive. In our scheme, the CPU passes the correction angle to the Cordic processor once for each substrate and then passes each coordinate pair for rotation as needed. As will be shown, the rotation time for the Cordic processor is comparable to the I/O cycle time of the CPU.

The Cordic algorithm and its adaptation to small-angle rotation

Volder[2] developed the Cordic algorithm to solve the trigonometric relationships associated with navigation. His approach solved the traditional trigonometric resolver equations with a system of shifts and adds. Daggett[3] demonstrated how Cordic could be used to perform decimal-to-binary and binary-to-decimal conversions. Walther[4] extended the Cordic technique to a variety of functions in addition to trigonometric ones. Steer[5] described a high-speed Cordic coprocessor—for finding sines and cosines—for a PDP-15 minicomputer. Haviland[6] proposed a complete CMOS VLSI Cordic processor that is as yet unavailable commercially.

The basic Cordic computing modes are rotation and vectoring. The rotation mode computes the new coordinates of a vector after rotation, using the original coordinates and the rotation angle. The vectoring mode computes, from the coordinates of the vector, the magnitude and angular argument of the original vector. Since rotation is the mode employed by our special-purpose Cordic processor, it is the one we will review here. Our development follows that of Volder,[2] with minor modifications.

Assume that a vector (R,θ) expressed in terms of its rectangular components X and Y is to be rotated through a specified positive angle A. The resulting components X' and Y' of the rotated vector are given by the relations

$$X' = X\cos A - Y\sin A \quad (1)$$

and

$$Y' = X\sin A + Y\cos A \quad (2)$$

Dividing both sides of Equations 1 and 2 by cosA yields

$$X'/\cos A = X - Y\tan A \quad (3)$$

and

$$Y'/\cos A = Y + X\tan A \quad (4)$$

The rotation may proceed in steps—for example, $A = A_1 + A_2$. If an arbitrary number of steps is allowed, the vector components at each step are given by

$$X_{i+1} = X_i - Y_i \tan A_i \quad (5)$$

and

$$Y_{i+1} = Y_i - X_i \tan A_i \quad (6)$$

where i represents the step number. Each new pair of vector coordinates, X_{i+1} and Y_{i+1}, is a function of each in the preceding step, as indicated in Equations 5 and 6. Cordic implements each step except the first (i.e., i = 1) by the approximation diagrammed in Figure 3. Each angular increment a_i is defined as

$$a_i = \tan^{-1} 2^{-(i-2)} \quad i > 1 \quad (7)$$

With this restriction, each step is simply a shift and add process. Following the notation of the figure, we can express the X_{i+1} and Y_{i+1} components in terms of the vector magnitude R_i at the ith step:

$$X_{i+1} = \sqrt{1 + 2^{-2(i-2)}}\ R_i \sin(A_i \pm a_i) \quad (8)$$

and

$$Y_{i+1} = \sqrt{1 + 2^{-2(i-2)}}\ R_i \cos(A_i \pm a_i) \quad (9)$$

By substituting Equation 7 into Equations 5 and 6 for $\tan A_i$, we can express X_{i+1} and Y_{i+1} in terms of X_i and Y_i:

$$X_{i+1} = X_i \pm 2^{-(i-2)} Y_i \quad (10)$$

and

$$Y_{i+1} = Y_i \pm 2^{-(i-2)} X_i \quad (11)$$

As indicated in Figure 3, the new vector is actually a binary shift along the tangent of the arc of the previous vector. The simultaneous addition/subtraction process represented by Equations 10 and 11 is termed cross-addition.

The expressions for X_{i+1} and Y_{i+1} are not perfect rotations, since the magnitude of the vector increases at each step by the value under the radical in Equations 8 and 9. When the process is restricted to n steps, the error is a constant K given by

$$K = \prod_{i>1}^{n} 1/\cos A_i = \prod_{i>1}^{n} \sqrt{1 + 2^{-2(i-2)}} \quad (12)$$

The first rotational step requires special treatment. It achieves a true 90-degree rotation with no magnitude error, as shown in

$$X_2 = -/+ Y_1 ; Y_2 = +/- X_1 \quad (13)$$

A rotation of any angle of up to ±180 degrees may be achieved by adding or subtracting each a_i in turn. Addition or subtraction is selected at each step to drive the rotation angle toward 0, as indicated by the sign of A_i. The final accuracy is $\geq a_n$. The complete set of basic equations is

$$X_{i+1} = X_i - \text{sign}(A_i) 2^{-(i-2)} Y_i \quad (14)$$

$$Y_{i+1} = Y_i + \text{sign}(A_i) 2^{-(i-2)} X_i \quad (15)$$

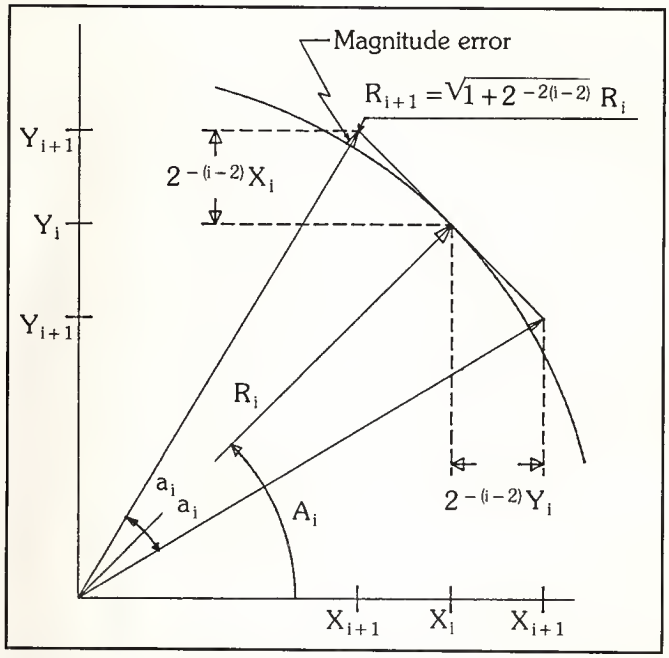


Figure 3. A typical Cordic rotation step.

Table 1.
Arc tangent radix (ATR) constants.

i	Arctan ½**(i−2)	Angle	Angle/180	ATR_i binary
1	infinite	90.0	0.5	0.100000000000000
2	1.0	45.0	0.25	0.010000000000000
3	0.5	26.565051	0.147583	0.001001011100100
4	0.25	14.036243	0.077979	0.000100111111011
5	0.125	7.125016	0.039583	0.000010100010001
6	0.0625	3.576334	0.019869	0.000001010001011
7	0.03125	1.789910	0.009944	0.000000101000101
8	0.015625	0.895174	0.004973	0.000000010100010
9	0.0078125	0.447614	0.002487	0.000000001010001
10	0.00390625	0.223810	0.001243	0.000000000101000
11	0.00195313	0.119056	0.000622	0.000000000010100
12	0.00097656	0.055953	0.000311	0.000000000001010
13	0.00048828	0.027976	0.000155	0.000000000000101
14	0.00024414	0.013988	0.000078	0.000000000000010
15	0.00012207	0.006994	0.000039	0.000000000000001

and

$$A_{i+1} = A_i - \text{sign}(A_i)\ a_i \quad (16)$$

Angles are represented as binary fractions of a half revolution. In this special radix representation, the a_i's are called arctangent radix constants, or ATRs. Their derivation and values to 16 bits are shown in Table 1.

```
                CORDIC TRANSFORMATION PROGRAM

significant digits:12
start rotation with step number: 1
X coordinate: 1643
Y coordinate: 848
Angle/180 (16 bit hex): 0444


step     Y register          X register          A register

 01  0.001101010000000  0.011001101011000  0.000010001000100
    +0.011001101011000 -0.001101010000000 -0.100000000000000
     ------------------  ------------------  ------------------
 02  0.011001101011000  1.110010110000000  1.100010001000100
    -1.110010110000000 +0.011001101011000 +0.010000000000000
     ------------------  ------------------  ------------------
 03  0.100110111011001  0.001100011011001  1.110010001000101
    -0.000110001101100 +0.010011011101100 +0.001001011100100
     ------------------  ------------------  ------------------
 04  0.100000101101101  0.011111111000101  1.111011100101001
    -0.000111111110001 +0.001000001011011 +0.000100111111011
     ------------------  ------------------  ------------------
 05  0.011000101111101  0.101000000100001  0.000000100100101
    +0.000101000000100 -0.000011000101111 -0.000010100010001
     ------------------  ------------------  ------------------
 06  0.011101110000001  0.100100111110011  1.111110000010101
    -0.000010010011111 +0.000001110111000 +0.000001010001011
     ------------------  ------------------  ------------------
 07  0.011011011100011  0.100110110101011  1.111111010100001
    -0.000001001101101 +0.000000110110111 +0.000000101000101
     ------------------  ------------------  ------------------
 08  0.011010001110111  0.100111101100011  1.111111111100111
    -0.000000100111101 +0.000000011010001 +0.000000010100010
     ------------------  ------------------  ------------------
 09  0.011001100111011  0.101000000110101  0.000000010001001
    +0.000000010100000 -0.000000001100110 -0.000000001010001
     ------------------  ------------------  ------------------
 0A  0.011001111011011  0.100111111001111  0.000000000111001
    +0.000000001001111 -0.000000000110011 -0.000000000101000
     ------------------  ------------------  ------------------
 0B  0.011010000101011  0.100111110011101  0.000000000010001
    +0.000000000100111 -0.000000000011010 -0.000000000010100
     ------------------  ------------------  ------------------
 0C  0.011010001010011  0.100111110000011  1.111111111111101
    -0.000000000010011 +0.000000000001101 +0.000000000001010
     ------------------  ------------------  ------------------
     0.011010001000001  0.100111110010001  0.000000000000111

rotated coordinates Kx,Ky: 2546.125,1672.125
```

Figure 4. Example of 12-step Cordic rotation. The original coordinates appear at top; the rotated coordinates are given at bottom. This example was produced by a 16-bit software simulator.

A rotation example generated by a 16-bit software simulation is shown in Figure 4. Note that the input angle must be converted to a fraction of 180 degrees in the same fashion as the ATRs. All computations are done with two's-complement arithmetic. Step 1 shows the initial values of Y, X, and A. The perfect rotation peculiar to this step implements Equation 13. Steps 2 through 12 implement Equations 14-16. At each step, the program drives the angle closer to zero by sensing the sign of A_i and adding ATR_i if it is negative and subtracting ATR_i if it is positive. The addend/subtrahend for X_i is Y_{i-1} right-shifted by i places; the Y_i case is complementary. Note that even though 16 bits are available, only twelve steps are used. This limits the accuracy to 12 bits plus sign. The rotated values must be divided by $K_{12} = 1.64676$ (Equation 12) to yield the correct results of X = 1546, Y = 1015.

Rotation of a vector through a given angle requires the sum of the incremental angles to be greater than or equal to the desired rotation as illustrated in

$$A <= a_1 + a_2 + a_3 + \ldots + a_n + a_n \quad (17)$$

This ensures convergence to within a_n of 0 in n steps. Cordic permits a maximum rotation angle of ±180 degrees. For a smaller maximum angle B, early steps may be eliminated, as shown below:

$$A <= a_1 + \ldots + a_n + a_n$$

$$B = A - a_1 - a_2 \ldots - a_j <= a_{j+1} + \ldots + a_n + a_n \quad (18)$$

```
                CORDIC TRANSFORMATION PROGRAM

significant digits:12
start rotation with step number: 6
X coordinate: 1643
Y coordinate: 848
Angle/180 (16 bit hex): 0444


step     Y register          X register          A register

 06  0.001101010000000   0.011001101011000   0.000010001000100
    +0.000001100110101  -0.000000110101000  -0.000001010001011
     ------------------  ------------------  ------------------
 07  0.001110110110101   0.011000110110001   0.000000110111001
    +0.000000110001101  -0.000000011101101  -0.000000101000101
     ------------------  ------------------  ------------------
 08  0.001111101000011   0.011000011000101   0.000000001110101
    +0.000000011000011  -0.000000001111101  -0.000000010100010
     ------------------  ------------------  ------------------
 09  0.010000000000111   0.011000001001001   1.111111111010011
    -0.000000001100000  +0.000000001000000  +0.000000001010001
     ------------------  ------------------  ------------------
 0A  0.001111110100111   0.011000010001001   0.000000000100101
    +0.000000000110000  -0.000000000011111  -0.000000000101000
     ------------------  ------------------  ------------------
 0B  0.001111111010111   0.011000001101011   1.111111111111101
    -0.000000000011000  +0.000000000001111  +0.000000000010100
     ------------------  ------------------  ------------------
 0C  0.001111110111111   0.011000001111011   0.000000000010001
    +0.000000000001100  -0.000000000000111  -0.000000000001010
     ------------------  ------------------  ------------------
     0.001111111001011   0.011000001110101   0.000000000000111

rotated coordinates Kx,Ky: 1550.625,1017.375
```

Figure 5. Example of small-angle Cordic rotation. Here, coordinate rotation starts at Step 6. The original coordinates appear at top; the rotated coordinates are given at bottom. This example was produced by a 16-bit software simulator.

$$B < = a_{j+1} + \ldots + a_n + a_n \quad (19)$$

The value of the constant K must be reduced by deleting factors associated with the eliminated angles. A small-angle Cordic example is shown in Figure 5. To produce a rotation of six degrees, the program starts at Step 6 because

$$6^\circ < a_6 + a_7 + \ldots + a_{12} + a_{12} = 7.11^\circ \quad (20)$$

The results are divided by the reduced $K_6 = 1.00260$ to yield X = 1547 and Y = 1015. This compares well with the results shown in Figure 4.

Design requirements for the Cordic processor

The largest substrate that our system must trim is 2×2 inches. Since the *x-y* table is moved in 0.5-mm steps, 4000 steps are needed to cover such a substrate. Consequently, 13 bits of resolution, including sign, are required. The coordinates derived will be too large by the K error factor. To correct for this, the system stores all coordinates in the lookup table as X/K and Y/K. (This is the technique proposed by Walther[4] and Steer and Penstone.[5]) However, these new values have built-in error since X/K and Y/K are not integer values, as are X and Y. Simulation results indicated that the error in the rotated values would be significant, suggesting that some form of postcorrection would be necessary. Postcorrection increases accuracy but requires a division operation on the result by K. We included postcorrection in our design.

The largest error angle we expect is 5 to 6 degrees. We consider any error beyond that a manufacturing defect. A 5- to 6-degree maximum error angle means that rotation can be started with Step 6. This is desirable for three reasons. First, it saves computation time. Second, as one can observe in Table 1, all ATRs in this angle range have five or more leading zeroes. Since ATRs are ROM-stored, leading zeroes can be hardwired to save storage. Third, starting rotation from Step 6 keeps the required resolution to 13 bits. Starting rotation from Step 1 would generate a magnitude error K of about 1.65; thus, the number of steps needed to cover the substrate would be 6600, not 4000, and 14 bits of resolution would be required. Starting rotation from Step 6 generates a K error of only 1.0026; thus, the number of steps required stays at 4000, and 13 bits of resolution are sufficient.

Other design criteria for our Cordic processor included

- a rotation time comparable to an I/O instruction cycle of the control computer, a 4-MHz Z80,
- the ability to build the processor with off-the-shelf parts,
- the ability to fit the processor on a standard IEEE 696 circuit board, and
- the ability to interface the processor to the Z80 as a group of I/O ports.

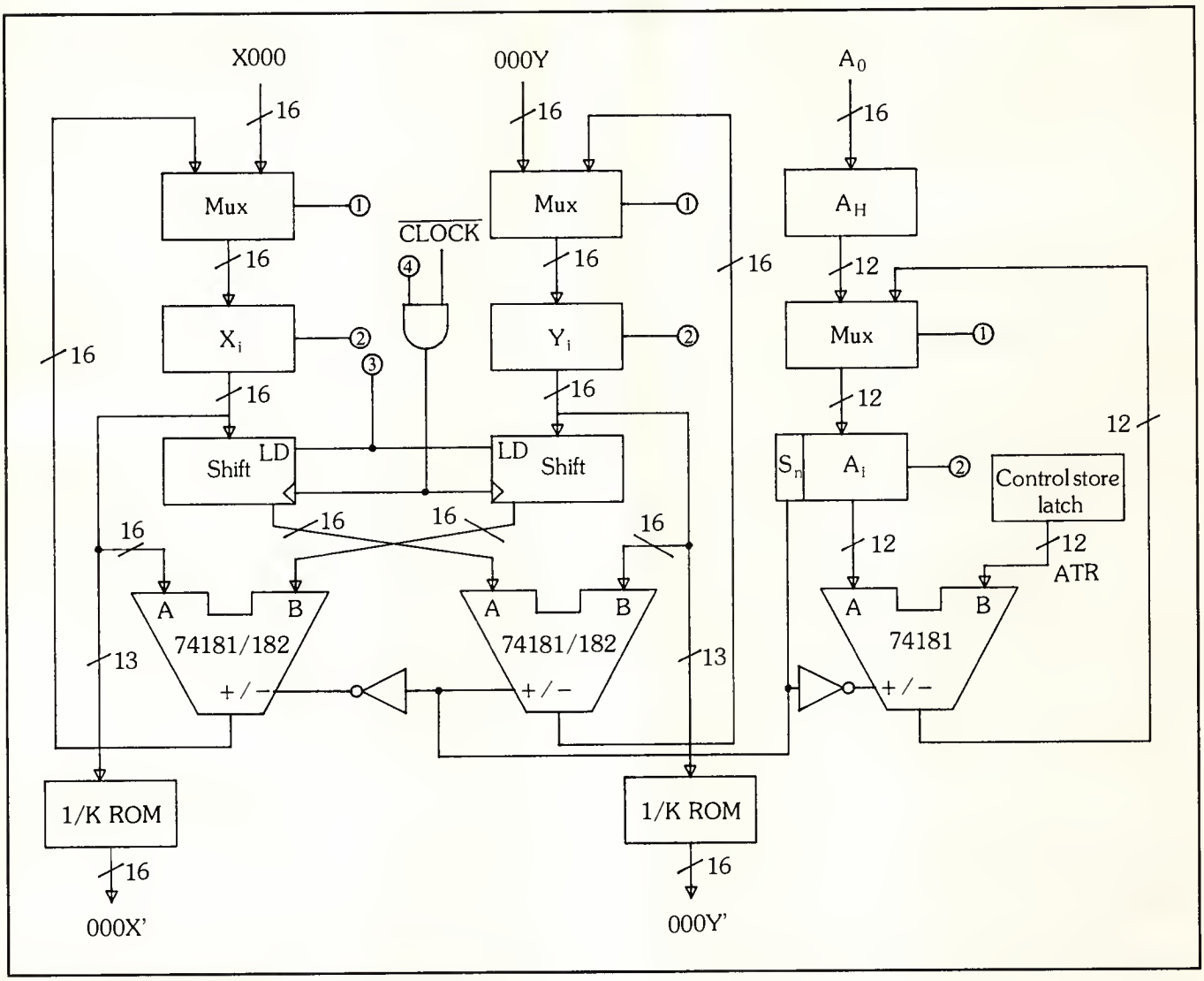


Figure 6. Block diagram of the Cordic processor.

Architecture of the Cordic processor

Figure 6 is a block diagram of the Cordic processor. Its architecture implements Equations 13 through 16. It passes values to and receives values from the laser processor by standard I/O techniques. The angle of rotation is loaded into its holding register, AH, where it is retained for all coordinate rotations of a particular hybrid microcircuit. Its input multiplexers are preconditioned to pass the angle from the AH register to an A_i register and latch the X and Y values directly into X_i and Y_i registers.

Rotation is initiated by an external START signal applied to the controller (see Figure 5 again). At each step, shifters are loaded with the current values of X_i and Y_i, then shifted right by the number of bits i required for that step. Adders perform cross-addition of the shifted and unshifted coordinates, and the result is latched. The latched values are X_{i+1} and Y_{i+1}, respectively.

As X_i and Y_i are iterated, the current angle A_i is also adjusted. The ATR is added to or subtracted from A_i to drive it closer to zero at each step. Whether it is added or subtracted is determined by the sign of A_i, which also selects addition or subtraction for the X and Y adders.

Two's-complement data are supplied as two 8-bit bytes. The angle data are a 16-bit representation of the error angle divided by 180. The coordinate data are a fraction of the full scale of the trimmer's range. Representation of the coordinate data requires only 13 bits but is expanded to 16 bits so off-the-shelf components can be used. The input path is wired to set the low-order bits of the coordinate data to zero and shift the data left. The additional three bits increase the accuracy of the intermediate steps beyond that attainable through simple truncation. Additional error is diminished through the use of von Neumann rounding (i.e., jamming the least significant bit with 1). While jamming, when used with mixed addition and subtraction, introduces

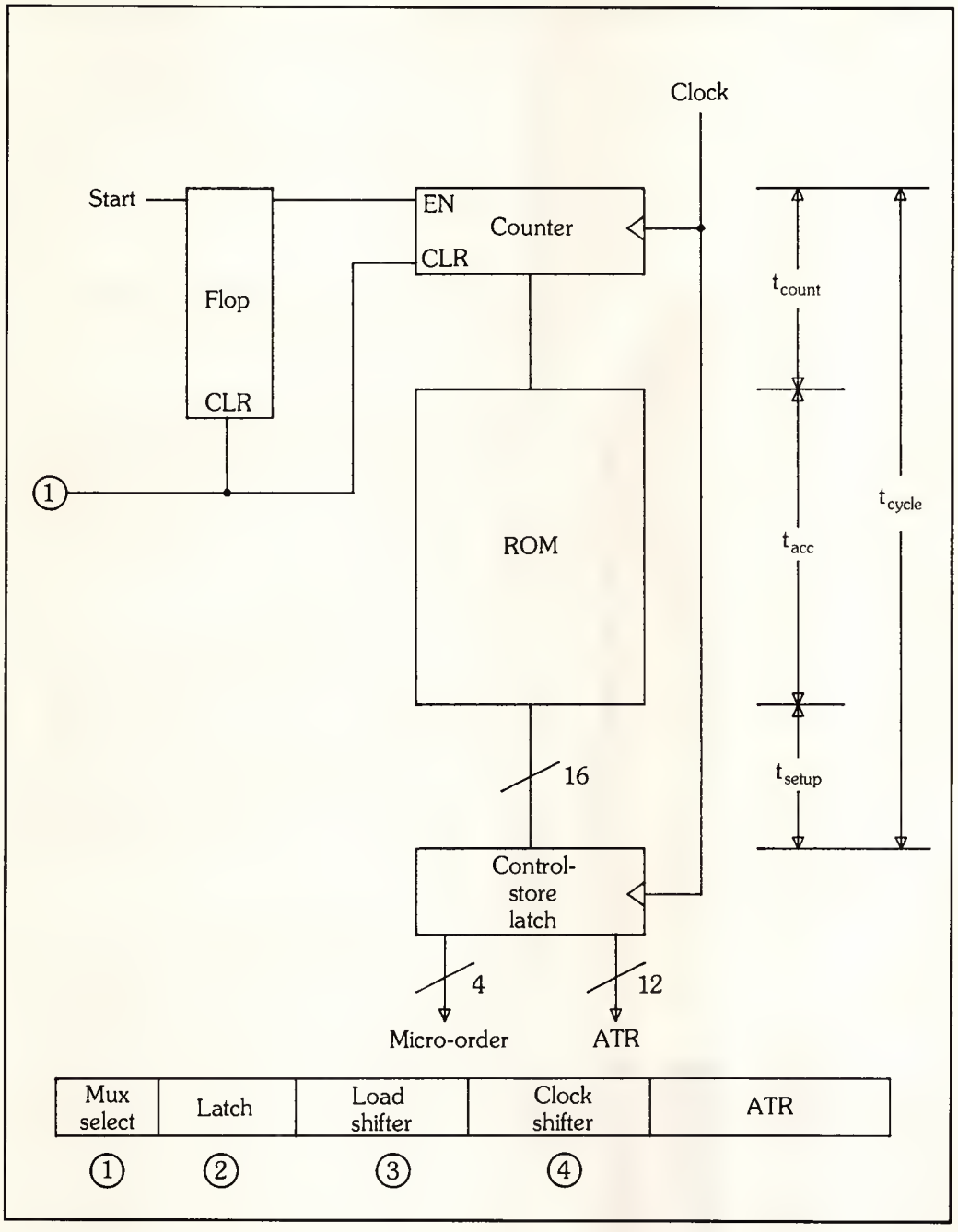


Figure 7. The microprogrammed controller for the Cordic processor.

more error for a single step than truncation does, it should have the effect of decreasing the total error.[7]

The results after n steps are too large by K. The coordinates form 13-bit addresses into ROM dividers. The outputs are corrected to 16-bit values with three leading zeroes by means of simple mapping.

We verified the architecture with a software simulator. Simulator outputs are shown in Figures 4 and 5. The details of each step help clarify the whole Cordic process. As discussed earlier, each rotation begins with Step 6 of the "full" rotation. The angles to be rotated have at least four leading zeroes when converted to a fraction of 180 degrees. Dropping four leading zeroes from the ATRs (Table 1) allows the angle data path to be reduced to 12 bits, saving one four-bit slice throughout. The final angle is not reported to the controller since it is guaranteed to converge to 0 ($< a_{12}$).

The controller is microprogrammed (Figure 7). Each control-store address contains an ATR field of 12 bits and a purely horizontal micro-order field of four bits. Upon START, the counter is enabled and rotation commences.

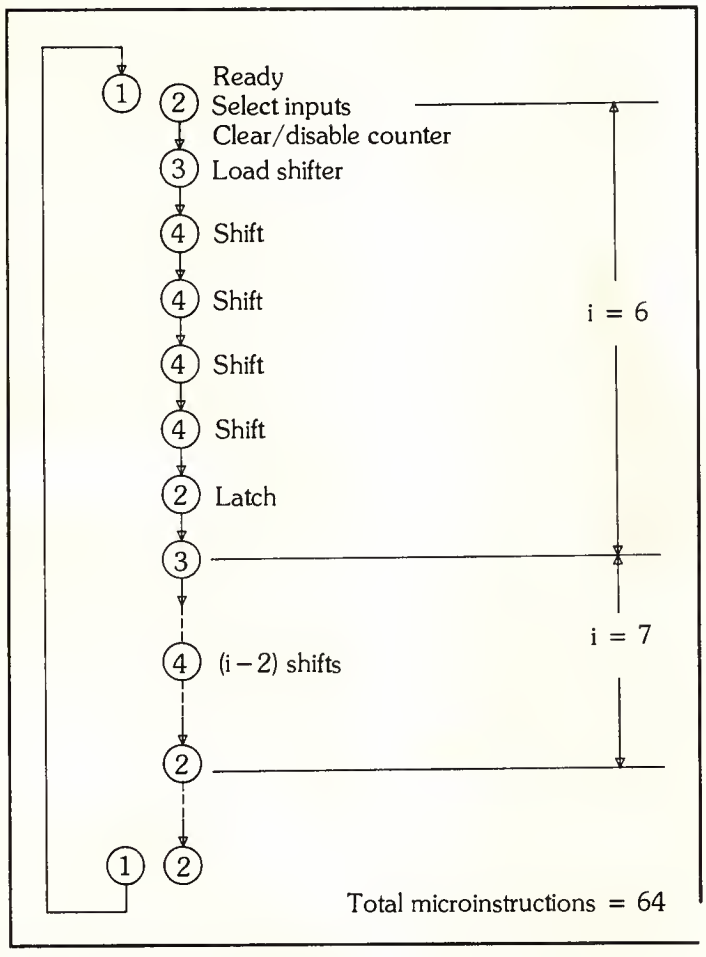


Figure 8. Flow diagram of the microprogram for the Cordic processor controller.

Table 2.
Micro-operations for the Cordic processor control unit.

Micro-order	Micro-operation
1.	Select initial x,y and angle inputs to the muxes.
1.	Stop counter (control-store memory address).
1.	Clear counter.
2.	Strobe registers.
3.	Load shifters.
4.	Enable clock to shifters.

The steps are shown in the flowchart in Figure 8. The microcode is strictly linear, with each step longer by one microinstruction (one shift) than the last. The micro-operations are listed in Table 2. In the final step, micro-order 1 disables the counter and resets it to zero for the next rotation. This same signal is passed back to the laser controller as a READY signal. The microprogram has a total of 64 steps.

Hardware implementation of the Cordic processor

The actual hardware implementation was driven by two requirements. First, the conversion time had to be comparable to the basic I/O instruction cycle of the laser controller, which is a 4-MHz Z80. That is,

rotate time $\cong$ 11 clocks * 250 ns = 2.75 μs.

Second, the whole processor had to fit on a standard IEEE 696 circuit board (5 × 10 inches).

The speed of the Cordic processor was determined by the cycle time of the controller. This is set by the sum of the address counter delay, the ROM access time, and the data register setup time (see Figure 7 again). The most suitable 512 × 8 PROM for the controller was the 82S147A, which has a maximum access time of 45 ns. The best data register was the 74F374, which has a setup time of 2 ns, and the best counter was the 74F163A, which propagates an output in 11 ns. Therefore, the maximum clock speed of the controller is 58 ns (17 MHz).

The longest processor data path delay is the sum of the add time, the mux propagation time, and the register setup time (see Figure 6 again). Three 74F181 ALUs, each with a 74F182 carry-lookahead unit, offer a maximum add/subtract time of 29.5 ns. A 74F257 propagates an output in 7 ns and a 74F374 register requires a 2-ns setup. These times add up to 38.5 ns and leave a margin of 19.5 ns. This time can be traded off for lower power consumption by substituting a 74LS257A for the 74F257, which raises the time to 49.5 ns. This is still well within the margin. The angle loop has at least two clock cycles in which to add or subtract an ATR, since there is no shift requirement. A lower-power 74LS181 ALU can be used in ripple mode for an add time of 78 ns. Of course, interpart propagation time cannot be ignored at these speeds. Assuming a worst-case distance of 12 inches on our 5 × 10-inch board and a propagation time of 2 ns per foot, we obtain an acceptable processor clock speed.

The final results must be divided by K. This is accomplished through a simple ROM mapping scheme. The 13 bits of X and Y each address a pair of 8K × 8 PROMs (82HS641's). The 16-bit quotients are available in 45 ns and the three unused leading bits are set to 0.

With 64 microinstructions to process for a rotation, the Cordic processor will be able to produce a result in 4.29 μs.

A very simple interface to the controller can be obtained by having the controller issue five no-ops of 1 μs each and then read the result.

Significant effort was expended in reducing shift time in earlier Cordic designs.[5,6] Examining the microcode (Figure 8) shows the reason. The shift length and therefore the total time grows with the number of steps. Only a simple shift register was needed in our design because of its overall speed. However, an alternative worth mentioning is shown in Figure 9. This is a combinational shift array built from programmable array logic. Each block shifts from 0 to 4 positions in 25 ns. Connected as shown, a Cordic step counter could select a 0- to 12-bit shift in 75 ns. This would require a slower clock, but the overall savings would offset this. Each step would have three microinstructions, for a total time of

(6 steps) * (3 microinstructions) * 75 ns
+ 45 ns = 1.40 μs,

which is three times faster than our implementation. However, board space constraints ruled out this approach.

The second requirement for the hardware implementation of the Cordic processor was that it fit on a standard IEEE 696 circuit board. This requirement is just met by our design, which employs 14 packages of 24 pins each, 14 packages of 20 pins each, 21 packages of 16 pins each, and four packages of 14 pins each.

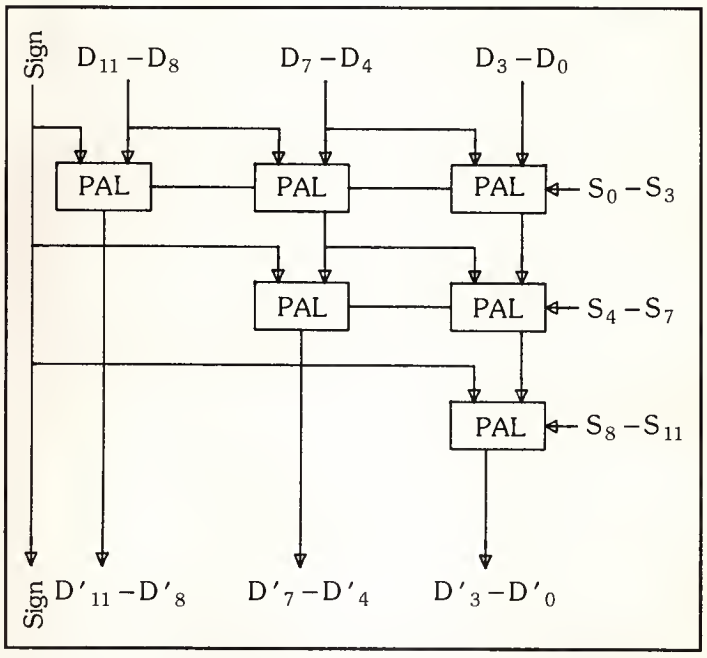


Figure 9. Programmed array logic implementation of a combinational shift network. A Cordic step counter can be connected to this network.

As we pointed out earlier, a typical 30-resistor substrate requires that 300 coordinate pairs be rotated to their true positions. Our original software rotation routine, written in assembler, is a virtual, 16-digit, decimal machine that implements the trigonometric functions. A second software rotation package, written in compiled Basic and using standard library functions, was also implemented as a potential replacement for the original routine. Times to rotate a representative coordinate pair and to perform the 300 coordinate rotations required for a typical 30-resistor substrate are shown in Table 3 for these two routines, for an Am9511A coprocessor, and for the Cordic processor.

The table shows that the Cordic processor performs the required 300 coordinate pair rotations in less time than the other three rotation techniques take for a single rotation. The 9511A implementation offers a simpler hardware alternative to rotation than the Cordic processor, but it doesn't provide the advantages of the Cordic technique—faster rotation time and less driver software.

Table 3.
Comparison of single and multiple (i.e., 300) coordinate rotation times for various rotation techniques.

Rotation technique	Rotation time for a single coordinate	Rotation time for a 30-resistor hybrid (300 rotations)
Virtual machine	845 ms	4.22 min
Compiled Basic	178 ms	43.4 s
Am9511A coprocessor	8.66 ms	2.60 s
Cordic processor	4.29 μs	1.29 ms

The time savings that can be achieved by incorporating the Cordic processor into a laser trimming system become evident when we consider a typical trimming production run of 40 hybrid microcircuit resistors. With software-based rotation, the system takes 13 minutes to trim a hybrid microcircuit,[1] resulting in an overall trim time of 8.66 hours for a 40-circuit production run. With the Cordic processor, the system takes 8.78 minutes to trim a hybrid, resulting in an overall trim time of 5.85 hours for the 40-unit run. (Little improvement beyond 8.78 minutes is attainable, since the bulk of this remaining time represents the time needed for manual setup.) The time savings of 2.8 hours are significant enough to justify the Cordic approach to rotation.

Other applications of the Cordic processor

The Cordic processor design can be easily adapted to perform the complete Volder rotation cycle, which covers a maximum angle of ± 180 degrees. The major hardware change consists of expanding the angle processing loop to a full 16 bits. This affects the control-store ROM by requiring 16 bits for the ATRs. The 90-degree rotation in Step 1 is affected by the latching of X_i and Y_i in the shift registers, by the clearing of the X_i and Y_i registers, and by the adding/subtracting from 0. Clearing the two registers requires another micro-order from the control store. This bit and the increase in ATR length requires the addition of another 512 $\times$ 8 ROM. The remaining steps are unchanged. The microprogram is modified to include the previously omitted rotation steps. The number of microinstructions increases from 64 to 83, and there is a corresponding increase in computation time.

A function complementary to rotation is vectoring,[2] and the Cordic processor can be adapted to perform it as well. In vectoring, a zero angle is loaded and a coordinate pair is iterated, with the length of the iteration dependent on the sign of Y_i. When $Y_i = 0$, the angle register contains the original angle of the vector and X_i contains the magnitude. To support vectoring, the Cordic processor must be able to accept from the control processor a control signal that indicates whether rotation or vectoring is to be performed, and it must have logic to select the correct sign bit. The microprogram for vectoring is the same as that for rotation.

Our design deals with the circular coordinate system. It can perform various functions in the linear and hyperbolic systems as well.[4] The essential hardware remains unchanged.

We have described a hardware implementation of the coordinate rotation, or Cordic, algorithm. The hardware, a special-purpose processor, is used for the laser trimming of hybrid microcircuits. We have adapted the Cordic algorithm to the small angles encountered in this application, reducing computation time and parts count. We utilized a microprogrammed design and verified it by means of software simulation.

We found prescaling of the *x-y* coordinate values an unsatisfactory solution to the magnitude errors inherent in the Cordic algorithm. We included postcorrection in the design, using ROM lookup tables. This also saved word width by limiting the magnitude expansion.

The Cordic processor occupies one IEEE 696 card and provides a rotated vector to the control computer in only 4.29 μs, whereas the software method needs 845 ms to provide such a vector.

References

1. J. A. Howard and E. B. Croson, "Microprocessor Control of a Microcircuit Laser Trimming System," *Proc. First Int'l Symp. on Mini- and Microcomputers in Control*, San Diego, Calif., Jan. 1979, pp. 195-200.
2. J. E. Volder, "The Cordic Trigonometric Computing Technique," *IRE Trans. Electronic Computers*, Vol. EC-8, No. 3, Sept. 1959, pp. 330-334.
3. D. H. Daggett, "Decimal-Binary Conversion in Cordic," *IRE Trans. Electronic Computers*, Vol. EC-8, No. 3, Sept. 1959, pp. 335-339.
4. J. S. Walther, "A Unified Algorithm for Elementary Functions," *AFIPS Conf. Proc.*, Vol. 38, 1971 SJCC, pp. 379-385.
5. D. G. Steer and S. R. Penstone, "Digital Hardware for Sine-Cosine Function," *IEEE Trans. Computers*, Vol. C-26, No. 12, Dec. 1976, pp. 1283-1286.
6. G. L. Haviland and A. A. Tuszynski, "A Cordic Arithmetic Processor Chip," *IEEE Trans. Computers*, Vol. C-29, No. 2, Feb. 1980, pp. 68-79.
7. J. J. F. Cavanagh, *Digital Computer Arithmetic*, McGraw-Hill, New York, 1984, p. 430.

T. W. Curtis is a project engineer at the Microelectronics Laboratory of the Pacific Missile Test Center in Point Mugu, California. Since assuming his current position in 1980, he has been involved in computer applications, special-purpose architectures, and telemeter system design. He is presently designing custom VLSI circuits. He received the BS degree in EE and CS from the University of Califor-

nia, Berkeley, in 1975, and the MS degree in electrical and computer engineering from the University of California, Santa Barbara, in 1985. A member of Tau Beta Pi and Eta Kappa Nu, Curtis holds a Navy Invention Disclosure.

Paul D. Allison works in the Missile Systems Test Branch of the Weapons Evaluation Division of the Pacific Missile Test Center, where he is involved in the application of automatic test equipment to Navy missile systems. He received the BS degree in mathematics from Pittsburg State College, Pittsburg, Kansas, in 1969, and the BS degree in electronics engineering from the University of Kansas in 1975. He is currently doing graduate work at the University of California, Santa Barbara.

James A. Howard has been a professor in the Department of Electrical and Computer Engineering at the University of California, Santa Barbara, since 1967. His research interests include digital systems design, microcomputer systems and applications, computer architecture, and real-time computing. He received BS, MS, and PhD degrees in engineering in 1956, 1959, and 1967, respectively, all from the University of California, Los Angeles.

Questions about this article can be directed to Curtis at the Microelectronics Laboratory, Bldg. 609, Code 1061.1, Pacific Missile Test Center, Point Mugu, CA 93042.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Interest Card.

High 156 Medium 157 Low 158

MicroStandards

P1296: The Interprocessor Communication Standard

Michael D. Rap
R. Scott Tetrick
Intel Corporation

The P1296 specification defines a functionally partitioned architecture in which the main system bus —the Parallel System bus—is optimized for communication between application processors. The local bus of each processor, on the other hand, serves primarily as an execution environment, providing the CPU with access to local memory and peripheral devices.

In March 1986 the Institute of Electrical and Electronic Engineers approved the Project Authorization Request for P1296. Among the objectives of this proposed standard were:

- Definition of a 32-bit, synchronous standard bus specification that would not be specific as to vendor or processor.
- A standard intended for general-purpose applications to optimize block transfers, including protocol for message passing. For real-time applications the bus should provide a means of ensuring an upper limit to message delivery time.
- A standard oriented toward multiple processor modules in a functionally partitioned configuration and heterogeneous processor types in the same system.
- A bus interface optimized to take advantage of VLSI.

An address space dedicated to intermodule communication—message space —is an essential part of the P1296 architecture. When CPUs need to communicate with each other, they talk through message space instead of through shared memory, as is typically the case in traditional architectures. It is this message passing protocol and its integration into the bus interface silicon—the Message Passing Coprocessor—which represents the P1296 architecture's most significant contribution to the design of efficient, high-performance multiprocessor systems.

No other bus architecture, defacto or public, has fully addressed the problem of communication between modules. Earlier bus specifications defined only the physical interface to a standard system bus. This required implementers to develop a method of interprocessor communication (IPC). But since no standard for IPC existed, each implementer's IPC differed. As a result, though all boards in a system met electrical specifications, communication between them was still not guaranteed.

Most IPC implementations were based on shared memory structures. In these structures, commands and data were passed between system modules. The memory was either globally accessible or dual-ported (addressable by the on-board CPU and also the system bus). Dual-ported implementations became preferred because the number of system bus accesses could be reduced. These shared data structures were, in truth, special-purpose memory that could not be protected with standard capabilities and could not be used for other purposes, such as data storage. This left implementers with limited flexibility.

In addition as system bus structures standardized only the electrical characteristics, protocols, implemented in software, had to be created for IPC. These protocols required knowledge of the system's addressing capabilities and the special shared memory structures on each module in the system. The client (a CPU requesting services) needed to be able to interrupt a server (a provider of services), and vice versa. This software ran on CPUs as an application program or driver requiring extensive operating system support and intimate knowledge of the hardware. Both modules, client and server, had to change when the underlying hardware was changed or upgraded.

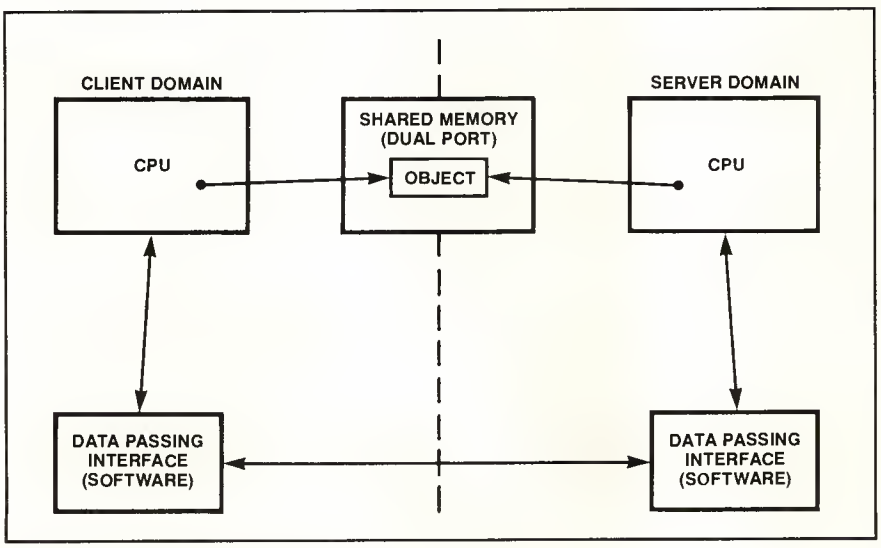


Figure 1. Passing data by reference.

The traditional approach to IPC

The traditional approach to interprocessor communication is based on two essential components—some kind of shared memory and one of two software protocols (either pass-by-reference or pass-by-value).

0272-1732/86/0600-0072$01.00 © 1986 IEEE

A typical communication sequence between two intelligent boards proceeds as follows (see Figure 1):

1. Dedicated memory space is allocated between the client (the CPU requesting services) and the server (the board providing the services) at configuration time. This must be done when the system software is generated.
2. In the dedicated memory space, the client places information for the type of command to be performed. If significant amounts of data are associated with the command (larger than the command block size), the command typically includes a pointer to the data. This is often referred to as passing the data by reference.
3. The client activates an "interrupt" to the server. In response to the interrupt, the server reads the command information from the dedicated command area.
4. At the completion of the command operation, the server activates an interrupt to the client. The client can now use the results of the operation for further processing.

The pass-by-reference method offers high performance but is very difficult to extend beyond two or three processors. This difficulty arises out of the fact that traditional interprocessor communication centers around the need for address aliasing. The address used to access the dedicated memory in dual-ported implementations as a system resource may not be the same as that used by the local processor. To communicate the location of the data in memory space, all CPUs in the system must have the capability of translating their addresses to those used by any other CPU. As the number of CPUs increases, this address translation task, or address aliasing, becomes time consuming and degrades system performance. Because the memory used for communication is globally accessible, it cannot be protected from errant operations, and conventional memory protection methods cannot be used.

In contrast, a pass-by-value model involves passing the data itself between the communicating modules. This sequence of events proceeds as follows (see Figure 2):

1. Dedicated memory space is allocated between client and server at configuration time. This must be done when the system software is generated.
2. The client copies the data from its local memory to the dedicated shared memory and signals the server.
3. The server copies data from the shared memory to its local memory and signals the client.
4. Steps 2 and 3 repeat until the data transfer is completed.

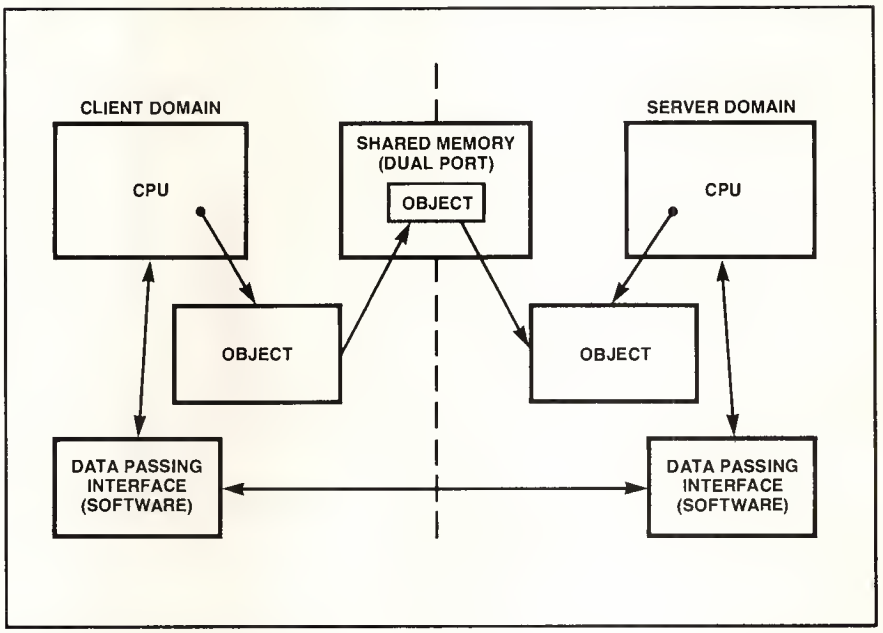


Figure 2. Passing data by value.

This solves a number of problems inherent in the pass-by-reference method, such as software complexity, address aliasing, and the conflict with memory protection. However, because the pass-by-value model requires movement of the entire data structure, it significantly reduces system performance in multiprocessor systems.

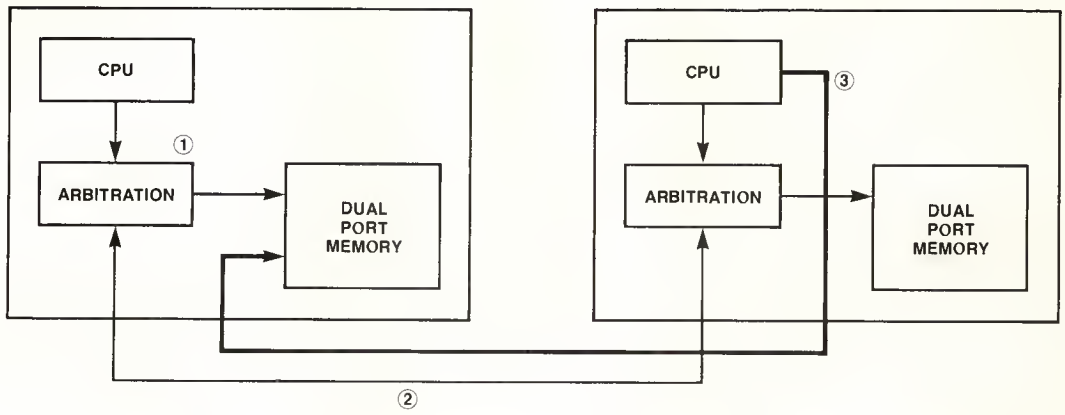
To maintain high performance when large amounts of data are to be transmitted and to solve the address aliasing problem, some designers find it necessary to store the data in dual-ported memory. If the data is not in globally accessible memory, it must first be copied from local memory before the command to the server can be processed.

Figure 3 shows the effect of moving data from one board's dual-ported memory. In this example, Board A has the data stored in its dual-ported memory. No on-board memory is really dual-ported; there is logic on the board to arbitrate between requests from the local CPU and the system bus. When an access from a board on the system bus occurs, it has the effect of coupling all of the buses on the boards involved: the local bus of the accessing board, the system bus, and the local bus of the dual-ported board. No new operations on any of these buses can be initiated until the current operation is completed. The speed of the dual-ported memory access is set by the slowest of the three buses. In addition, hardware delays, caused by arbitration for each of the involved buses, increase the access time of every transfer.

As shown in Figure 3, the net transfer rate of the dual-ported memory operations is less than one megabyte per second. If any of the CPUs involved has a lower bandwidth or smaller data path, i.e., eight or 16 bits instead of 32, system performance drops even further.

A further challenge of traditional multiprocessing architectures is the number of dedicated interrupt lines required to signal between CPUs. As processors are added to the system, a fixed number of interrupt lines limits performance. If there are N processors in a system, the number of interrupt lines required for each processor to interrupt any other processor in the system is $N \times (N-1)$. With four CPUs the number of required interrupt lines already exceeds the number provided by traditional buses today—there are only seven on IEEE P1014 (VMEbus) and eight on the IEEE 796 (Multibus I) system bus.

The P1296 architecture overcomes this limitation by using "virtual" interrupts. Message space transfers on the PSB con-



Bus Number	Action	Max. Bandwidth	Bandwidth Used	Bandwidth Available
1	Held, Waiting For Completion Of Dual Port Operation	16 MB/Sec	≤1 MB/Sec	0 MB/Sec
2	Involved In Transfer	40 MB/Sec	≤1 MB/Sec	0 MB/Sec
3	Involved In Transfer	1 MB/Sec	≤1 MB/Sec	0 MB/Sec

Figure 3. Effect of shared memory architecture on system performance.

tain an eight-bit field for both the source and destination of the message. One address is reserved for broadcast operations, where all boards responding to message space operations are addressed. The eight-bit source and destination values emulate 255 dedicated input interrupt lines to each board.

The new standard for IPC: Message Passing

The message passing protocol of the P1296 architecture, implemented by bus interface hardware, offers a unique and higher performance alternative to these traditional methodologies. The local CPU simply instructs the interface hardware to send a message, passing data by reference. The interface hardware then executes the pass-by-value transmission of the data over the system bus without further CPU intervention.

The message passing protocol of P1296 standardizes the format for communicating messages (data or interrupts) in a multiprocessing system. Messages are either unsolicited (such as command or interrupt) or solicited (such as the

Table 1.
Unsolicited message format.

	Address/Data Lines (bit positions)			
	31..24	23..16	15..8	7..0
Address			Source Address Byte 2	Destination Address Byte 1
Command	Undefined with valid parity		Type Specific Byte 4	Type Byte 3
Data (optional)	Data Byte 3 Byte 8	Data Byte 2 Byte 7	Data Byte 1 Byte 6	Data Byte 0 Byte 5
Data (optional)	Data Byte 7 Byte 12	Data Byte 6 Byte 11	Data Byte 5 Byte 10	Data Byte 4 Byte 9
Data (optional)	Data Byte 11 Byte 16	Data Byte 10 Byte 15	Data Byte 9 Byte 14	Data Byte 8 Byte 13
Data (optional)	Data Byte 15 Byte 20	Data Byte 14 Byte 19	Data Byte 13 Byte 18	Data Byte 12 Byte 17
Data (optional)	Data Byte 19 Byte 24	Data Byte 18 Byte 23	Data Byte 17 Byte 22	Data Byte 16 Byte 21
Data (optional)	Data Byte 23 Byte 28	Data Byte 22 Byte 27	Data Byte 21 Byte 26	Data Byte 20 Byte 25
Data (optional)	Data Byte 27 Byte 32	Data Byte 26 Byte 31	Data Byte 25 Byte 30	Data Byte 24 Byte 29

Note: The length may vary in 4 byte increments (minimum 4 bytes).

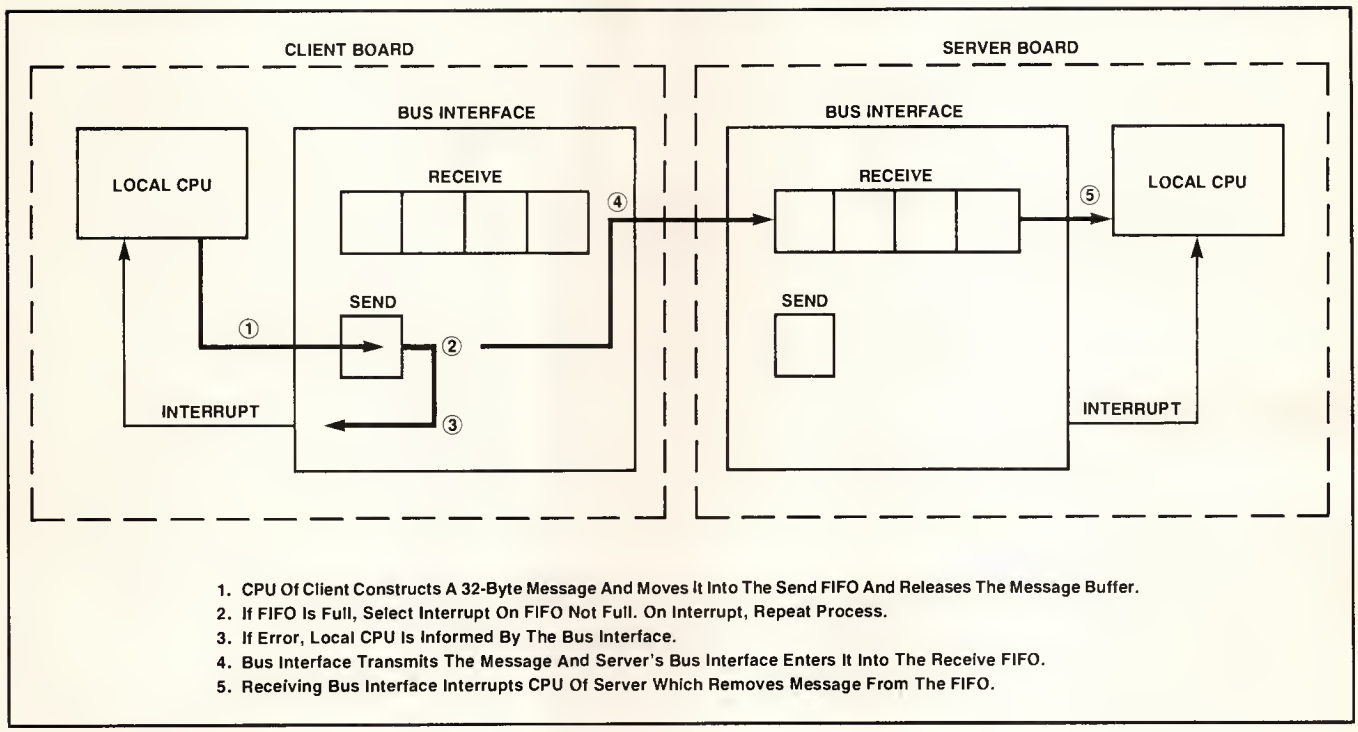


Figure 4. Sending an unsolicited message.

Table 2.
Solicited message format.

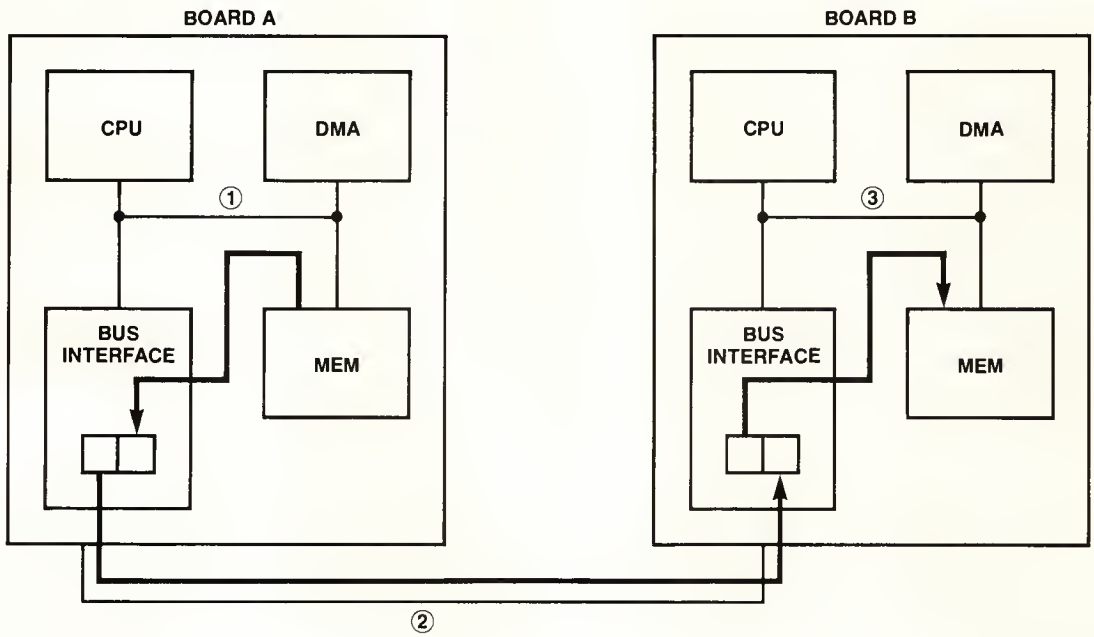
	Address/Data Lines (bit positions)			
	31..24	23..16	15..8	7..0
Address			Source Address Byte 2	Destination Address Byte 1
Command	Undefined with valid parity		Type Specific Byte 4	Type Byte 3
Data (optional)	Data Byte 3 Byte 8	Data Byte 2 Byte 7	Data Byte 1 Byte 6	Data Byte 0 Byte 5
Data (optional)	Data Byte 7 Byte 12	Data Byte 6 Byte 11	Data Byte 5 Byte 10	Data Byte 4 Byte 9
Data (optional)	Data Byte 11 Byte 16	Data Byte 10 Byte 15	Data Byte 9 Byte 14	Data Byte 8 Byte 13
Data (optional)	Data Byte 15 Byte 20	Data Byte 14 Byte 19	Data Byte 13 Byte 18	Data Byte 12 Byte 17
Data (optional)	Data Byte 19 Byte 24	Data Byte 18 Byte 23	Data Byte 17 Byte 22	Data Byte 16 Byte 21
Data (optional)	Data Byte 23 Byte 28	Data Byte 22 Byte 27	Data Byte 21 Byte 26	Data Byte 20 Byte 25
Data (optional)	Data Byte 27 Byte 32	Data Byte 26 Byte 31	Data Byte 25 Byte 30	Data Byte 24 Byte 29
Data (optional)	Data Byte 31 Byte 36	Data Byte 30 Byte 35	Data Byte 29 Byte 34	Data Byte 28 Byte 33

Note: The length may vary in 4 byte increments (minimum 4 bytes).

movement of data). See Tables 1 and 2 for the structure of both types of message packets.

These two distinct types of communication have different characteristics. Unsolicited messages are unpredictable events, frequently resulting in interrupts and requiring quick response by the receiving CPU. Solicited messages, on the other hand, can be predicted and negotiated, have no size limitations, and should take up as little system bandwidth as possible.

To send an unsolicited message, the sending CPU follows a simple procedure (see Figure 4). The local CPU writes the unsolicited message into a dedicated transmit FIFO and signals the completion to the bus interface. The bus interface then sends the contents of the FIFO as a single packet (uninterrupted bus operation). If the packet cannot be received at this time—indicated by a Negative Acknowledgment (NACK) by the receiver—the bus interface, and not the local CPU, automatically retries the access. The local processor need be informed only when the retry limit has been reached.



Bus Number	Action	Max. Bandwidth	Bandwidth Used	Bandwidth Available
1	DMA Loading Solicited Output Buffer From Memory	16 MB/Sec	1 MB/Sec	15 MB/Sec
2	Bus Interface On Board A Unloading Buffer; Bus Interface On Board B Loading Buffer	32 MB/Sec	1 MB/Sec	31 MB/Sec
3	DMA Unloading Solicited Input Buffer To Memory	1 MB/Sec	1 MB/Sec	0 MB/Sec

Figure 5. Message passing architecture decouples buses and increases performance.

From the local CPU the bus interface hardware can now be designed to take full advantage of processor speeds. Accesses to the bus interface registers can be performed without the need to insert processor wait states, even for high speed CPUs. Accesses to the FIFO registers can be performed using block/string move instructions, for example. When the entire message has been written to the bus interface, the local CPU writes the command register to complete the operation. This signals the bus interface that the message can be transmitted and the bus interface immediately requests the use of the Parallel System bus. After loading the message into the FIFO, the bus interface transmits the message (at the full bandwidth of the PSB), leaving the CPU free to perform other operations.

Receiving a message is similarly easy. When a message is received for the bus interface's message address, it stores the entire message in one of the receiving buffers. If the receiving buffers were previously empty, the bus interface can interrupt the local CPU. The message can be read from the Receive FIFO by the local CPU for further processing.

The capability of passing small amounts of data in the unsolicited messages can markedly improve system performance. For example, a serial controller board interrupts a CPU board to indicate that a character has been typed at one of its terminals. In a dedicated interrupt line environment, the CPU, after receiving the interrupt, would have to read the status information from the board to indicate which serial channel requires service, then read the serial channel to determine the type of service required, and finally, read the character from the serial channel. In the shared memory environment, the single interrupt is followed by three additional bus operations conducted by the CPU. In a

message passing system, all of that status information, and even the byte of data, can be included in the interrupt message. Computational and system bus resources are freed in message passing systems, increasing the available processing power of the system.

To move data between CPUs, the P1296 specification defines special versions of the unsolicited messages called Buffer Request and Buffer Grant. The board with the data to be sent transmits a Buffer Request message, in the same manner as a normal unsolicited message. Specific data fields, for example, the number of bytes to be sent, are embedded within the Buffer Grant message.

Twenty-four additional bytes of data may be sent in the Buffer Request message. The receiving bus interface stores the information in its receiving buffers. The local CPU removes this message and determines that it is a Buffer Request by its type number. The CPU then attempts to allocate a block of memory of the requested number of bytes, via whatever local means available. If the CPU cannot complete the attempt, it returns a Buffer Reject message to the sender. If the CPU is able to allocate that amount of memory, the CPU transmits a Buffer Grant to the sender. Embedded in the Buffer Grant message is a duty cycle parameter to prevent buffer overruns.

Both client and server can program their dedicated hardware, such as DMA, for data movement, no longer requiring CPU intervention. At the completion of the data transfer, the CPUs on both client and server can be interrupted by the bus interface.

Hardware, software performance improved

For both unsolicited and solicited messages, buffers within the bus interface decouple the buses of the client and server. With unsolicited messages, the bus interface takes care of arbitrating for the PSB, performing the message transfer, and, if necessary, performing any NACK recovery. The CPU is only required if the message is not successfully received by the destination. With solicited messages, the buffers serve to pipeline the transfer of data between client and server. (See Figure 5).

For the sending bus interface, a buffer can be filled at the same time as one is transmitted over the PSB. A receiving bus interface can receive data simultaneously from the PSB and empty the other buffer onto the local bus. None of the buses are coupled for data movement. While the end-to-end transfer time is improved over dual-ported systems, the available processing power within the system is increased significantly. In our example the transfer speed is greater than 1M byte/sec., but the sending CPU, with a local bus bandwidth of 16M bytes/sec., still has 15M bytes/sec. of bandwidth available for other functions.

In the shared memory case discussed earlier, the sending CPU performance dropped to that of the receiver. It is also important to note the available bandwidth on the system bus. In the traditional architecture, the system bus bandwidth dropped to equal that of the slowest board involved in the transfer, or to 1M byte/sec. Because of the use of buffers in data movement, the data bandwidth available (after all address information has been removed) is still 31M bytes/sec. By integrating the message passing functions into message passing silicon, designers are insulated from slow CPUs or data paths of less than 32 bits. Rather than synchronize for each data transfer, as in the dual-ported approach, the bus interface synchronizes only once per packet (32 bytes), yielding a marked improvement in performance. Because the bus interface is optimized for data transmission, it has an effective throughput that greatly exceeds that of today's CPUs or DMA controllers. All transfers over the PSB occur at the full 32M-byte/sec. data rate.

Since message passing architectures do not require shared memory on the bus, full memory protection capabilities can be implemented. Each CPU in the system manages only its own local memory. In fact, operating systems running in a local environment are completely invisible to other boards in the system. Initially the interrupt structures of these operating systems must change to support the message passing functions. It is quite possible to mix operating systems such as DOS, Unix, and iRMX all within the same system, communicating via message passing.

The use of a dedicated message address space eliminates the address aliasing problem. Interprocessor communication in the system does not rely on shared memory because all information is passed to the receiving board. Pointers to the information are never passed. No longer do CPUs within a system require a memory map for each processor. Reliance on a dedicated address space eliminates dual-ported memory configuration and simplifies the job of system software generation.

The P1296 Message Passing Standard

Indeed the P1296 Message Passing Standard facilitates the design of multiprocessor systems because it allows for a partitioning of the system hardware and software. System software no longer coordinates shared data structures. Instead, the various processing units of the system communicate through standardized software procedures, that is, virtual interrupts and negotiated movement of data via messages. This, in turn, allows the software on a given board to be totally independent of the software environment of the other boards in the system.

Because interprocessor communication is integrated into the bus interface, and message passing serves to connect the various system functions, designs of both hardware and software have greater flexibility. It becomes much easier to upgrade a specific subsystem because of the limited impact on the software and execution environment of other pieces of the system. This opens the door for future technology and greatly eases the design of multiprocessing systems.

Additional information on the P1296 specification may be obtained by contacting Steve Cooper, Intel Corporation, 5200 NE Elam Young Parkway, Hillsboro, OR 97123.

MicroCourses

Buying, Installing, and Maintaining a Telecommunication System, July 14-16, Cambridge, Massachusetts; August 18-20, San Francisco, California, $895; **Computer Networks,** July 28-30, San Francisco, California, $485; **Data Networks: Management, Operation, and Control,** August 18-20, Boston, Massachusetts; $895. Contact Technology Transfer Institute, 741 Tenth Street, Santa Monica, CA 90402.

SNA, July 14-16, Washington, DC, $795; **Local Area Networks,** July 23-25, Washington, DC, $795; **Capitalizing on Telecommunications,** July 24-25, Chicago, Illinois, $995. Contact Systems Technology Forum, 10201 Lee Highway, Suite 150, Fairfax, VA 22030; (703) 591-3666 or (800) 336-7409.

Data Center and Telecommunication Facilities Design Seminar, July 17-18, Dallas, Texas; $350. Contact The Crosby Group, Inc., 7400 East Orchard Road, Suite 170, Englewood, CO 80111; (303) 796-7477.

Introduction to Local Area Networks, July 21-22, $650; **Local Area Network Applications and Implementation,** July 23-25, $750; **Internet Systems and Protocols,** July 28-August 1, $950; **Digital Telephony,** August 4-8, $950; **Electro-optics, Fiber Optics, and Lasers for Nonelectrical Engineers,** September 8-10, $750; Washington, DC. Contact The George Washington University, Continuing Engineering Education, Washington, DC 20052; (202) 676-6106 or (800) 424-9773.

Micro-Mainframe Links, July 22-23, San Francisco, California, $795; **Local Area Networks: Selection and Implementation,** August 21-22, Montreal, Canada; $795. Contact Digital Consulting Associates, Inc., 6 Windsor Street, Andover, MA 01810; (617) 470-3870.

Contemporary Data Communication Networks: Planning, Management and Computer-Based Design, August 4-8, Ann Arbor, Michigan; $700. Contact University of Michigan, College of Engineering, 300 Chrysler Center/North Campus, Ann Arbor, MI 48109-2092; (313) 764-8490.

Information Display Systems Engineering, August 4-8, Los Angeles, California; $995. Contact UCLA Extension, 10995 Le Conte Avenue, Los Angeles, CA 90024; (213) 825-1295 or 825-3344.

Surface Mount Design and Manufacturing Short Course, August 7-8, Chicago, Illinois; $500. Contact AWI Surface Mount Technology, 558 Oakmead Parkway, Sunnyvale, CA 94086; (408) 720-8860.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Interest Card.

High 195 Medium 196 Low 197

Correction

In Part II of "A High-Level-Language Programmable Controller" by Daniel A. Mange (April 1986, pages 47-63), there were several typographical errors:

- On page 54, in Figure 2, the labels "$BUS_{7:4}$" and "$BUS_{3:0}$" at the left end of the horizontal bus should be interchanged.
- On the same page, in the same figure, the labels "$MC_{23:0}$" and "$MC_{23:20}$" under the shaded arrows at far right should be interchanged.
- On page 57, in Table B, "$\bar{R} \oplus S$" in the line for OP_6 : EXOR should be "$R \oplus S$."

Also, on page 62, the first part of Table 9b was omitted. The omitted portion appears at right:

Comments	LLL code = $ROM_{31:0}$	HLL mnemonics
	——	PROCEDURE WHILE
Transfer of i into temporary register	10 30 50 9F	DO TEMP1 ← $RAM_{3:0}$ (= $i_{3:0}$)
Addressing RAM with actual LLL address	14 40 40 5B	DO ADR ← LPC (= ADRWHILE)
Writing IF x_i ELSE.. instruction	18 – -- 00 11 50 20 5B 12 40 50 5B	DO REG(8) ← 00 DO $RAM_{15:12}$ (ADRWHILE) ← REG(8) (= 0) DO $RAM_{11:8}$ (ADRWHILE) ← TEMP1 (= $i_{3:0}$)
Push ADRWHILE on stack for further calculation of IF x_i ELSE ADRENDWHILE + 1	14 40 10 5B 13 40 40 5B 10 40 10 CB	DO ADR ← SP (stack pointer) DO $RAM_{7:0}$ (SP) ← LPC (= ADRWHILE) DO SP ← SP – 1
Increment actual LLL program address	10 40 42 C3 A- -- -- --	DO LPC ← LPC + 1 END PROCEDURE

MicroLaw

by Richard H. Stern/Law Offices of Richard H. Stern/2101 L Street NW, Suite 800/Washington, DC 20037

Reverse engineering of chips

Part I: The legislative background

Distinguishing between legitimate reverse engineering of a competitor's chip layout and "chip piracy" under the new Semiconductor Chip Protection Act is both important and perplexing. In this two-part series I recount the events and decisions leading to the 1985 passing of the reverse-engineering provisions of the new law and present an illustrated case example for your consideration. After reading Part I, interested readers may wish to send in further examples. Please do so; I will be happy to comment on these cases in the December issue.

Probably, reverse engineering and second-sourcing were the most controversial issues before Congress during its consideration whether to enact the Semiconductor Chip Protection Act (SCPA) (Public Law 98-620). Congress finally did enact the chip protection law, but before it did it repeatedly amended the reverse-engineering aspects of the bills. In doing so, it may have created a murky line between legitimate reverse engineering on the one hand and illegal chip piracy on the other hand. That result stems in large part from the compromises that Congress made among competing interests and interest groups. In part, however, the lack of clarity is inherent in the subject matter—it is part of the territory.

Content of the chip law

The SCPA creates a new form of intellectual property right: the topography or layout of semiconductor chip products. The SCPA defines semiconductor chip products to include integrated circuits, but the definition appears to be broad enough to cover related products, such as power transistors and solar cells, if they are manufactured by IC technology (that is, photolithography or "masking" processes).

The owner of rights (called "mask work rights" under the SCPA) is ordinarily the semiconductor manufacturer or other employer of the personnel responsible for a chip layout. Interesting questions arise as to who owns the mask work rights in automatically placed and routed chips, but that issue will not be considered here except insofar as it relates to reverse engineering. The owner's rights last for 10 years after registration or first commercial exploitation, whichever comes first.[1]

There is no examination system for mask works, as there is under the patent system. Instead, the layout is simply registered in the Copyright Office, and the courts are expected to sort out any legal problems about validity or technical merit, if and when litigation occurs over an infringement of mask work rights. At that time the court must decide whether the layout was both (1) "original," and (2) not "staple, familiar, or commonplace." The SCPA requires both conditions to be met before a court enforces mask work rights against an alleged infringer.

The owner of mask work rights has the legal right to exclude other persons from reproducing the mask work in any form, such as mask, pattern-generation tape, or chip; from distributing or importing chips embodying the mask work; and from causing other persons to engage in any of the foregoing acts. A person who violates any of the owner's exclusive rights is an infringer of these rights. An infringer may be liable to the owner for damages for diverted sales of chips, may be required to pay the mask work owner any profits made from the sale of infringing chips, and may be enjoined against further acts of infringement.

There are limitations on the rights of a mask work owner. For example, the mask work owner cannot control what customers do with chips that they have purchased from the owner. Furthermore, an innocent purchaser of an infringing chip may be wholly exempt from liability or may be entitled to dispose of the chip (such as by putting it into equipment and selling it) upon payment of a reasonable royalty.

The most important limitation is the right of reverse engineering, provided under section 906(a) of the SCPA. It is a complete defense to a charge of infringement of mask work rights that the challenged conduct falls within the reverse-engineering exemption. Reverse engineering is probably the most important defense in cases of mask work infringement. Often, it will be the only possible defense; more often than not, some kind of reverse-engineering argument is available. If the reverse-engineering defense is established, the defendant is completely immunized from liability for infringe-

ment of mask work rights; there will be no damages, award of profits, injunction, nor any other relief granted in favor of the mask work owner.

Congressional considerations

As originally introduced, the Senate and House chip bills had no reverse-engineering provisions; they were drafted as additions to the copyright law. Congressional sponsors perhaps overconfidently gave assurances that the "fair use" exemption found in the Copyright Act would fully protect the industry practice of reverse engineering. A number of witnesses (ADAPSO, CCIA, NEC) expressed concern, however, that the courts would *not* interpret the Act and the fair use doctrine in accordance with these assurances.

At the same time the publishers' association (AAP) protested that bringing reverse engineering under the fair use doctrine would lead to a general and undue expansion of that doctrine, and thus permit an erosion of the rights of owners of traditional copyrights. No one opposed continuing the semiconductor industry's established practice of reverse engineering; the only question was whether it was enough to rely on statements in the legislative history or whether instead an express statutory guarantee was necessary. The latter view prevailed. However, there was a great deal of difficulty in securing unanimity on the wording of the reverse-engineering provisions, and they were rewritten repeatedly until the eve of passage of the SCPA.

The SCPA does not limit legitimate reverse engineering to abstract study; competitive chips can be sold.

The main purpose of the revisions to the earlier version was to make it explicit that legitimate reverse engineering was not limited to abstract study and could properly lead to manufacture and sale of competitive chips. Some very late legisla-

Industry concept of reverse engineering

What the semiconductor industry calls "reverse engineering" reflects the highly competitive customs of the industry and its traditional emphasis on competition in the form of continued technological improvement. Semiconductor chip manufacturers often make themselves "second sources" for products that another chip manufacturer has pioneered, with or without the pioneer's consent. However, the second firm usually does not simply duplicate the first chip without improving or reworking it. Typically, the second manufacturer attempts to improve such things as signal/noise ratio and thermal stability, to decrease chip (die) size to conserve "silicon real estate," and to decrease the number of masks and wafer-fabrication steps.

Extensive testimony before both houses of Congress explained the industry concept of reverse engineering, and the aspects of it that the witnesses felt were important to preserve and carry forward in the SCPA. Semiconductor industry representatives testified that it is an established industry practice to make photographs of one another's chips to analyze them and design similar chips, which are then sold in competition against the earlier chips. Such analysis permits the design of a second chip with the same electrical and physical characteristics as the first chip (a chip that is "form, fit, and function compatible") and therefore one fungible, or interchangeable, with it in the marketplace. This practice creates second sources of supply for chips and is considered fair competition—when the conduct is on what the industry considers the right side of the line between piracy and legitimate reverse engineering.

The right side of the line is that where the second chip manufacturer's photography and reproduction of the layout of the first chip is for the purpose of study and analysis, rather than just to appropriate the first chip manufacturer's labor and expense in creating and laying out the first chip. In both situations the two chips are "substantially similar" in the copyright law sense. In the case of reverse engineering, however, the second firm has invested substantial toil and expense in analysis.

In contrast, in the case of chip piracy the second chip is "the mere result of plagiarism accomplished without such study or analysis." The fact that the second semiconductor chip company invests its own substantial labor and expenditures in developing the second chip—rather than simply trying to cut corners and avoid the expense of independent chip-layout work—is perceived as critically important. As one witness (Tom Dunlap of Intel, basing his remark on a letter submitted by Les Vadasz of Intel) said:

> When there is a legitimate job of reverse engineering, there is a very big paper trail [documentary evidence of reverse engineering in a company's records and files], there's computer simulations, there's all kinds of time records, people who have spent an enormous time understanding and figuring out how to make that design.

Another difference is that the work recorded by the "paper trail" left by reverse engineering almost inevitably leads to a second chip layout that is not substantially identical to the first, unless the second firm's engineering efforts are completely ineffective.

The industry concept of reverse engineering has sometimes been equated to the copyright law's concepts of "fair use" or of copying "idea" rather than "expression." But those are concepts that apply better to books and similar traditional subjects of copyright; they translate imperfectly to semiconductor chips and other industrial property.

The Senate Report states, in explaining the difference between reverse engineering and piracy, that "the bill is directed at the appropriation of substantial parts of the drawings embodied in the masks and chips, when that is done to take free advantage of the first comer's great costs in developing the layout of the chip." This statement far more accurately captures the "anti-misappropriation" flavor of the semiconductor industry's concept of reverse engineering than do the "fair use" or "idea-expression" analogies from copyright law.

tive history explaining these changes confirmed that SCPA section 906(a) permits a chip resulting from reverse engineering to be "substantially similar" to the pioneer mask work, but not be "substantially identical" (namely, a virtual photocopy). In the course of so providing, however, Congress went further and required that the product of reverse engineering itself be "an original mask work" if it is to escape liability. This last change is nowhere explained in the legislative history, and the final language of section 906(a) about "original mask work" appears in none of the earlier published versions of the bill.

The unofficial legislative history of section 906(a) sheds some light on this otherwise mysterious provision. An unofficial, revised draft of the chip bill that was circulated for comment and suggestion in July and August of 1984 provided a version of section 906(a)(2) that immunized from infringement liability the conduct of a person who reproduced the work "in a substantially similar, but not substantially identical, mask work embodied in a semiconductor chip product that the person distributes." This provision reflected an effort by various industry representatives to clarify and emphasize the difference between the reverse engineering of a semiconductor chip product under the SCPA and fair use of a book under copyright law. The language "in a substantially similar, but not substantially identical, mask work" troubled many of those involved, however, because it seemed too complicated a notion for statutory language (as contrasted with legislative history language).

A less complicated substitute for this phrase was then suggested, "in an original mask work prepared to be commercially exploited," and it was combined with the language quoted above to produce the final version of SCPA section 906(a)(2). The concept of permitting the second semiconductor chip product to be "substantially similar" but not "substantially identical" to the first semiconductor chip product was then moved from the statute to the legislative history explaining it (where, it was felt, more complexity was permissible). Without any detailed public explanation, the requirement of "an original mask work" thus came into the statute.

As enacted section 906(a) provides:

> section 905 [concerning the exclusive rights of a mask work owner], it is not an infringement of the exclusive rights of the owner of a mask work for
>
> (1) a person to reproduce the mask work solely for the purpose of teaching, analyzing, or evaluating the concepts or techniques embodied in the mask work or the circuitry, logic flow, or organization of components used in the mask work; or
>
> (2) a person who performs the analysis or evaluation described in paragraph (1) to incorporate the results of such conduct in an original mask work which is made to be distributed."

An overview of reverse engineering under the SCPA

SCPA section 906(a) permits competitors of a mask work owner to reproduce the mask work for reverse-engineering purposes, even though SCPA section 905(1) gives mask work owners the exclusive right to reproduce the layout. The first step of the reproduction is to photograph the layout of the original chip. An enlargement of the photograph may be used to prepare a composite drawing. The same pictorial material may be digitized and stored in a tape. Each of these actions is a reproduction of the protected mask work, which ordinarily violates the SCPA. All these actions are permissible, however, if part of legitimate reverse engineering.

The competitor will then analyze the foregoing material and the chip itself. The competitor will ascertain the circuit schematic and the logic flow within the chip; it will infer the physical and electrical specifications of the chip from study of the product itself and from the needs of its circuitry. Eventually, the competitor will combine the results of these reverse-engineering efforts with its own engineers' engineering efforts to yield a new and possibly improved version of the chip. The question then becomes one of whether the resulting version of the chip has an infringing layout.

A spectrum of similarity in layout may be relevant in judging chip layouts in reverse-engineering situations. The spectrum could run as follows: total identity, substantial (although not total) identity, substantial similarity (without substantial identity), insubstantial similarity, no similarity. In the first two parts of the spectrum the second chip manufacturer is always liable to the first under the SCPA, since reverse engineering is a defense only when the resulting chip is not so close to the original that the two are substantially identical. In the last two parts of the spectrum the second chip manufacturer is never liable to the first under the SCPA, since the defendant always prevails if the two chips are not even substantially similar.

The middle of the spectrum is the interesting part. When the two chips are only substantially similar, reverse engineering can be a valid defense. Then, the mask work owner prevails in the lawsuit unless the second manufacturer proves reverse engineering (or some other defense).

When the second chip is in the middle part of the similarity spectrum, manufacture of the chip is a violation of the reproduction right and its sale is a violation of the distribution right, unless the second chip satisfies the following two additional criteria:

- it is "the product of substantial study and analysis," as evidenced ordinarily by a paper trail showing considerable expenditure of time and money; and
- it embodies an original mask work.

Of the three criteria that the defendant must satisfy to establish the reverse-engineering defense—lack of substantial identity, substantial study and analysis, and embodying an original mask work—the first two are more or less objective and quantitative while the third is more qualitative and subjective. Instead of measuring points of similarity or the amount of time and money spent, the third criterion poses the question: Did the result of the defendant's investment, toil, effort, study, and so on result in a work that is not only not identical to the plaintiff's but also sufficiently different from the plaintiff's to be an "*original* mask work"? We have seen, above, how the language got into the SCPA, but it is quite different to explain what the language means, particularly in the context of a reverse-engineering dispute.

The word "original" is not defined in the SCPA. Moreover, the Act uses the word three times—in SCPA sections 902(b)(1), 902(b)(2), and 906(a)(2)—each time in a slightly different context and sense. The first use is part of the standard for registrability of a layout; that sense appears to correspond to the meaning of "original" as it is used in the copyright law, which is merely failure to plagiarize or copy the work from another person. Possibly in this use the

word carries some added freight of novelty or of being first in time. In SCPA section 902(b)(2) protection is barred for semiconductor designs that are merely variations of staple "designs, combined in a way that, considered as a whole, is not original." In that context, "original" definitely has some connotation of novelty, over and above mere failure to copy from another person.

When SCPA section 906(a)(2) limits the reverse-engineering privilege to "an original mask work" that incorporates the results of analysis and evaluation of the earlier mask work, "original" clearly means something different. The word

"Original" must mean the reverse-engineered work is different in some way.

cannot mean mere lack of copying, for the context of reverse engineering presupposes some amount of copying; similarly, it cannot mean first in time. The "original mask work" of SCPA 906(a)(2) must mean something else. It must, for lack of any other possibility, mean being different in some amount from the original work on which the reverse-engineered work is based. Unfortunately, just what degree of novelty or creativity all of this necessitates is left unclear by the mysterious legislative history of SCPA section 906(a)(2).

Probably the best analogy is suggested by several cases relating to copyrights in "derivative works." In several cases involving toy products based on prior works, which were either in the public domain or were originally (that is, initially) created by a licensor of both parties, the court held that such a derivative work could support a new copyright only if the new work was more than trivially different from what came before and from which it was derived. SCPA section 906(a)(2) appears to impose the same rule on reverse-engineered mask works, but in the context of avoiding infringement liability rather than that of gaining copyright protection. The later layout, therefore, probably should be at least more than trivially different from the earlier one. (That does not necessarily mean that the second mask work would qualify for registrability under the SCPA, for that is not what section 906(a)(2) requires.)

This is an unusual noninfringement test for intellectual property law. Having any merit or qualifying for any kind of intellectual property protection is usually neither a necessary nor a sufficient condition to avoid liability for infringement of rights in an earlier work. An improvement patent is likely to infringe any "dominant" patent to which it is "subservient," even though it is an improvement on it, and a derivative-work copyright often cannot be exploited without infringing the work from which it is derived.

But the case of reverse engineering is different from the usual rules of intellectual property law. SCPA section 906(a)(2) takes the unusual step of making this particular kind of derivative work, a reverse-engineered mask work, free of subservience to the earlier work. The later mask work is wholly immune from liability for infringement of earlier mask work rights if it qualifies as the result of reverse engineering.

It is doubtless this unusual rule of law that leads to the result that section 906(a) requires some slight technical merit in the otherwise infringing chip to excuse it from infringement liability. The requisite merit, however, is only a small quantum of originality or technical advance, and not necessarily enough creative merit to transcend the staple and commonplace.

Applying the reverse-engineering defense

The legislative history of SCPA section 906(a) asserts that rarely will it be difficult to distinguish between chip piracy and legitimate reverse engineering, because most actual situations tend to fall into one of those two polar categories rather than into any "gray area" between them. The Senate Report on the bill summarized testimony to this effect in terms of two factors.

First, it is uneconomical for a pirate to copy only part of a chip and then spend his own money to design the rest; a strategy of wholesale appropriation is more sensible, from the pirate's standpoint. Second, the integration of the parts of a chip tends to discourage copying a part of a chip and combining the part with something else, because the combination is unlikely to function properly. Accordingly, the Senate Report concluded, most actual situations in this field are either at one end or the other of the spectrum that runs from clear copying to clearly legitimate reverse engineering. (Some observers, however, believe that this view is overly optimistic.)

Of course, that most actual situations in semiconductor industry settings are clear cases does not mean that most litigation situations will be as clear. Litigation tends to result from the minority of close cases rather than from the majority of clear cases. Nevertheless, uncertainty should be greatly reduced, the Senate Report states, by reliance on and use at trial of proper documentary evidence. This documentary evidence is the "paper trail" that legitimate reverse engineering leaves behind but chip piracy does not. The House Report concurred in the "evidentiary importance of the 'paper trail' of legitimate reverse engineering that helps distinguish it from mere piracy." Probably, the "paper trail" is the most important item of evidence in a reverse engineering case.

The "paper trail" is not the only evidence to be considered on reverse engineering. Testimonial evidence from expert witnesses is also important. Such testimony would help explain the significance of the paper trail or that of its absence. Such testimony would also cover the importance or lack of importance of the obvious or subtle changes that the defendant made in its version of the mask work. Finally, expert testimony may be important to explain why the second-source chip must adhere to all of the specifications established by the earlier chip, to be compatible in form, fit, and function, and why it is therefore true from a marketing standpoint that "better is worse" for the second-source chip.[2]

No one item of testimonial or documentary evidence is necessarily decisive. In the aggregate, however, properly selected evidence may paint a very persuasive picture of legitimate reverse engineering or wanton and blatant chip piracy. In general terms, the following are some of the kinds of evidence that properly would be presented and considered in deciding a reverse-engineering case:

What is a chip's "paper trail"?

The "paper trail" of reverse engineering is not quite the same as that of auditing and accounting, although it is in many ways similar. The concept is explained and described in the Senate Report by a quotation from an expert in chip technology, Les Vadasz of Intel, a quotation which the Senate said it agreed with and adopted "as a guide to its intent":

> Whenever there is a true case of reverse engineering, the second firm will have prepared a great deal of paper—logic and circuit diagrams, trial layouts, computer simulations of the chip, and the like; it will also have invested thousands of hours of work. All of these can be documented by reference to the firm's ordinary business records. A pirate has no such papers, for the pirate does none of this work. Therefore, whether there has been a true reverse engineering job or just a job of copying can be shown by looking at the defendant's records. The paper trail of a chip tells a discerning observer whether the chip is a copy or embodies the effort of reverse engineering. I would hope that a court deciding a lawsuit for copyright infringement under this Act would consider evidence of this type as it is extremely probative of whether the defendant's intent is to copy or to reverse engineer.

- What are the relative development costs of the two semiconductor chip products? The person-hour figures? The elapsed time from start to finish? A great disparity suggests misappropriation.
- Do the defendant's records show substantial expenditures for analysis, testing, computer simulations of the functioning of the chip, and other indications that the defendant independently exercised its judgment in determining the design it used?
- Are there common mistakes, unnecessary elements, arbitrary design choices? Is there the same "fix" to correct a prior error?[3] Such common features suggest piracy rather than reverse engineering.
- Does the second chip contain substantial design enhancements?

This discussion will continue in the next issue of *IEEE Micro* with a case history that may suggest either chip piracy or legitimate reverse-engineering. As mentioned earlier, additional examples submitted by interested readers will be considered for publication in the December 1986 issue. I invite such readers to furnish the appropriate supporting data (die photographs, schematics, etc.) as well as their comments on the diagnosis of reverse engineering or piracy.

References

1. This discussion is excerpted and adapted from the recent book, *Semiconductor Chip Protection,* by R. H. Stern, Harcourt Brace Jovanovich/Law & Business, 1986. The principal discussion of reverse engineering is in section 5.5 of the book. Supporting references for most propositions stated in this article are omitted but may be found in the book.

2. In defending against a charge of piracy, when there are at least many superficially similar aspects of the two products, and the circuitry did not compel particular design choices to enable the defendant to arrive at a working product, the defendant may need to have an expert explain to the court that the market compels "form, fit, and function compatibility" and that compatibility considerations require inclusion in the second chip of what seem to be flaws or arbitrary design choices in the original chip. For compatibility purposes "better is worse," for improvements may interfere with product interchangeability, and a non-interchangeable second product may be unsalable.

3. A well-publicized case of this sort occurred with the 8086 microprocessor, prior to the passage of the SCPA. At a very late stage in the development of the product, its designers decided that they had to change several 1's in the microcode of the ROM to 0's, apparently because they wanted to alter the instruction set. To remove these transistors entirely would have meant redoing several masks, with the attendant opportunity to make some mistakes. Therefore, the decision was made simply to remove the interconnections that "wired" these 1's into the circuit, so that the "floating," unconnected transistors would, in effect, become 0's. This meant changing only one mask—that for making the holes in an upper, insulating layer to permit aluminum to pass through them to make the interconnections. When another company then copied the 8086, it copied the several nonfunctional transistors along with the functional ones. See *Washington Post*, May 2, 1983, p. A1.

MicroReview

Editor: David L. Hannum/AT&T Information Systems

To Engineer Is Human

To Engineer Is Human—The Role of Failure in Successful Design, by Henry Petroski, St. Martin's Press, New York, 1985, $16.95. 247 pages; includes index and 132-item bibliography.

Reviewed by Joe Schallan, Contributing Editor

We live in a technological environment we largely trust. We expect neither bridges nor buildings to collapse. Great structures are an everyday part of our lives—we use them routinely.

But such was not always the case, as Henry Petroski emphasizes in *To Engineer Is Human*. In the 1880's, for example, one could reasonably argue that structural failure was the expected, even routine, fate of iron railroad bridges. Such failure—as well as locomotive boiler explosions, fire in the wooden cars of the age, and faulty steel rail—seized the public imagination. Rail travel was widely considered an inherently hazardous proposition.

Such failure was the most valuable school for structural engineers, however. The school of failure taught engineers more about the properties of materials, more about good design, than theory or textbook. And, most notably, it taught them more about their craft than their successes did.

Engineers have learned so well from failure that a major failure today is big news. We no longer expect bridges to collapse or buildings to fall in or spacecraft to explode. Such disasters are perceived as anomalous, as the sure result not of pushing the art past mapped frontiers but of gross incompetence or negligence. We grieve the lost lives, we search among the designers for the guilty. Yet these disasters serve the same function as the failures of an earlier era. Failure remains the engineer's best teacher, his best laboratory.

How does failure instruct the engineer? Petroski walks us through several famous failures and shows us how. The seemingly state-of-the-art suspension bridge over the Tacoma Narrows was broken by a moderate wind, and engineers learned that aerodynamic factors had to be taken into account during

Excerpts from the book

From the preface

"I believe that the concept of failure ...is central to understanding engineering, for engineering design has as its first and foremost objective the obviation of failure. Thus the colossal disasters that do occur are ultimately failures of design, but the lessons learned from those disasters can do more to advance engineering knowledge than all the successful machines and structures in the world. Indeed, failures appear to be inevitable in the wake of prolonged success, which encourages lower margins of safety. Failures in turn lead to greater safety margins and, hence, new periods of success. To understand what engineering is and what engineers do is to understand how failures can happen and how they can contribute more than successes to advance technology."

From Chapter 16, "Connoisseurs of Chaos"

"It is a great profession. There is the fascination of watching a figment of the imagination emerge through the aid of science to a plan on paper. Then it moves to realization in stone or metal or energy. Then it brings jobs and homes to men. Then it elevates the standards of living and adds to the comforts of life. That is the engineer's high privilege.

The great liability of the engineer compared to men of other professions is that his works are out in the open where all can see them. His acts, step by step, are in hard substance. He

design. After this well-publicized failure, no suspension bridge was undertaken without extensive wind-tunnel testing of models.

The 1981 collapse of the suspended walkways at the Kansas City Hyatt Regency taught the engineer another lesson—that even a sufficient design can be subverted during construction. The drawings showed connections adequate to hold two walkways suspended, one over the other, from the lobby ceiling. But a shortcut was taken during construction and the connections were altered. This change turned out to be critical, for it made the upper walkway barely able to support its own weight, much less that of a crowd gathered on it. The price of the shortcut was ultimately paid in 114 lives, but the lesson to engineers was immensely valuable.

Petroski conducts an anatomy of failure, one thought-provoking to both the engineer and those who care about engineers and their craft. He argues persuasively for the study of failure: "...nothing can erase an engineering disaster. Yet no disaster need be repeated, for by talking and writing about the mistakes that escape us we learn from them, and by learning from them we can obviate their recurrence."

Petroski is a structural engineer and hence his examples of engineering failure are largely those of failure of bridge, building, and vehicle. But his premise generalizes to all engineering. The computer engineer will find a great deal to think about here, especially since his designs are arguably the most complex structures ever undertaken by man. It would be valuable for a computer engineer to carry Petroski's analysis of failure into the microelectronic world, especially since computer engineering is not a mature discipline in the way structural engineering is, and, presumably, is still at the point in its development where failure is routine. What do computer engineers make of their mistakes? How can the study of those mistakes improve the craft?

The computer scientist will find a chapter addressed directly to his concerns. "From Slide Rule to Computer: Forgetting How It Used To Be Done" examines the effects of building engineering expertise into computer programs. Who should write these programs? Should engineers trust such programs? Should engineering programs, like engineers themselves, be registered?

Computer engineer and computer scientist alike should read this book, especially as chip and code enter tool, building, and vehicle. As our dependence on computers grows, so does our dependence on the skill and humility of their designers. Computer engineers and computer scientists would do well to create their own literature of failure and to read it closely.

> cannot bury his mistakes in the grave like the doctors. He cannot argue them into thin air or blame the judge like the lawyers. He cannot, like the architects, cover his failures with trees and vines. He cannot, like the politicians, screen his shortcomings by blaming his opponents and hope that the people will forget. The engineer simply cannot deny that he did it. If his works do not work, he is damned. That is the phantasmagoria that haunts his nights and dogs his days. He comes from the job at the end of the day resolved to calculate it again. He wakes in the night in a cold sweat and puts something on paper that looks silly in the morning. All day he shivers at the thought of the bugs which will inevitably appear to jolt its smooth consummation."

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Interest Card.

High 177 Medium 178 Low 179

FYI

The August issue of *IEEE Micro* features multiprocessing.

New Products

Editor: Kenneth Majithia/IBM Corporation

Send announcements of new microcomputer/microprocessor products, and products for review, to Managing Editor, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578.

System acquires, controls data 10,000 feet from PC

CyberResearch has announced the CyberDAS Remote Data Acquisition System, which uses its TeleDAS series of remote data acquisition and control modules. The system is made up of TeleDAS interface modules connected directly to transducers or other analog signals to be monitored, a twisted-pair network, a PC interface board, and control software. The modules can be located up to 10,000 feet from the PC; up to 32 modules can be connected to one twisted-pair line. According to the company, multiple twisted-pair lines can be used to accommodate hundreds of I/O modules.

The TeleDAS modules contain analog-to-digital converters with a resolution of one part in 50,000; a microprocessor; and RS-232, RS-422, or RS-485 serial communications interfaces. The microprocessor controls the A/D conversion, automatically calibrates the converter, and provides linearization and compensation functions for the transducer. Module output is automatically converted to standard engineering units.

Data acquisition and control software for the PC is included in the CyberDAS 2 system, providing an interface to the DAS network, data display, and advanced data analysis capabilities.

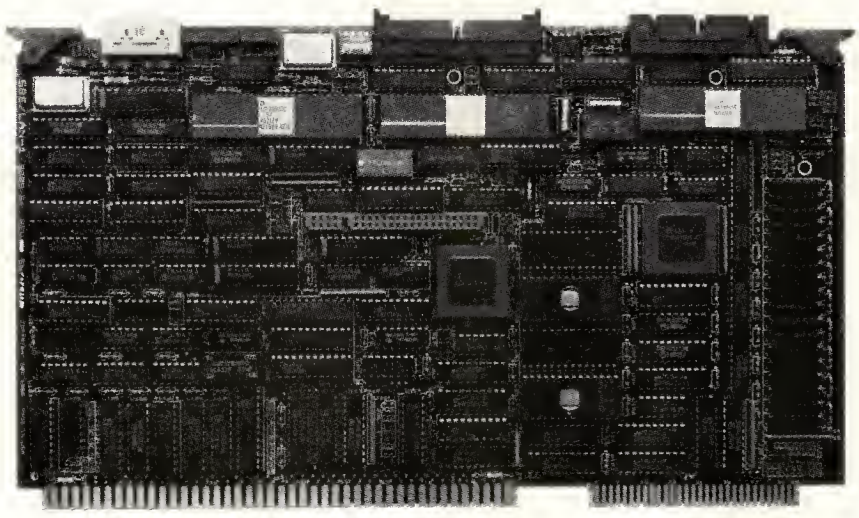
A complete CyberDAS 2 system ready for installation on a PC is priced at $1995. An additional charge of $295 is made for each TeleDAS module.

Contact CyberResearch Inc., at 5 Science Park Center, PO Box 9565, New Haven, CT 06536; (203) 786-5151.

Reader Service Number 10

Ethernet plug-in board supports Multibus systems

A plug-in Ethernet front-end processor for Multibus systems in local area networks has been announced by SBE, Inc. The MLAN-11 can be used as a typical Ethernet node processor functioning as a slave Ethernet Multibus controller, or it can be used as a packaged black box with a power supply to function as a stand-alone serial/parallel/SCSI gate to the Ethernet highway.



SBE's MLAN-11 plug-in board couples a 10-MHz 68000 microprocessor with a LANCE chip set to provide an Ethernet transfer rate of 10M bps. Protocols supported include X.25, SDLC, HDLC, Bisync, and Async, as well as NRZ, NRZI, and FM encoding and decoding.

Gateway tasks linking LANs and long-distance carriers, such as the X.25 wide area networks, can also be handled with the board. SBE provides X.25 packet-switched software that can be downloaded from a host or can reside on board in EPROM.

The MLAN-11 includes 128K bytes or 512K bytes of no-wait-state, dual-ported, dynamic RAM, with parity checking for detection of single-bit errors during memory access. A Mailbox Interrupt feature allows other Multibus masters to invoke various reset interrupts on the MLAN-11.

The MLAN-11 board is sold in lots of 100 for $1495 each.

For more information contact SBE, Inc., 2400 Bisso Lane, Concord, CA 94520; (800) 221-6458, or in California (800) 328-9900.

Reader Service Number 11

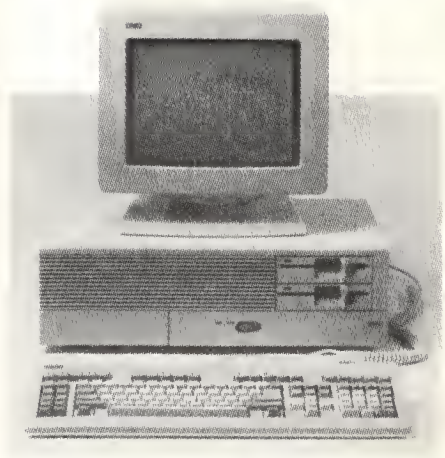
DOS helper offered for IBM PCs

Bourbaki, Inc., is offering its DOS-helper utility software to users of IBM PC and compatible systems. It replaces the usual DOS prompts of A> or C> with a full screen of information and menus that provide single-keystroke access to applications, sequences of operations, and DOS-level file commands.

1 dir Version 3.5 (pronounced "wonder") insulates users from complex commands and syntax, manages the details of handling computer files, and helps users create a quick-access menu system. The displayed list of files in the current directory can be sorted by name, extension, size, or date; users can choose whether or not to display hidden (special system) files or file creation dates. A Batch-Builder feature captures complete sequences of commands as batch (.BAT) files.

For price and additional information contact Bourbaki, Inc., 431 Main Street, Boise, ID 83702; (208) 342-5849.

Reader Service Number 12



NCR Corporation's PC8, an 80286-based, IBM PC AT-compatible unit, can be used as a stand-alone unit; a network file server; or a multiuser, multitasking system connecting up to 16 users. The suggested retail price in its basic configuration is $3795. Basic application software costs approximately $1600. For more information contact NCR Corporation, Public Relations, Dayton, OH 45479.

Reader Service Number 14

Scanner software reads dot-matrix output

DEST Corp. has enhanced the Text Pac character recognition and interface software for its PC Scan page reader for personal computers. Text Pac Version 2.0 enables the PC Scan to read output from dot-matrix printers. It also allows the scanner to automatically enter text in the formats of word-processing programs, including Microsoft Word, Word Perfect, WordStar, Displaywrite 3, and MultiMate Advantage.

The software permits the scanner to read documents printed in 10 and 12 pitch and the proportionally spaced type fonts commonly used in business documents from typewriters and printers.

Text Pac Version 2.0 is priced at $595.

DEST Corp. has headquarters at 1201 Cadillac Court, Milpitas, CA 95035; (408) 946-7100.

Reader Service Number 13

Supermicro added to Xelos computer line

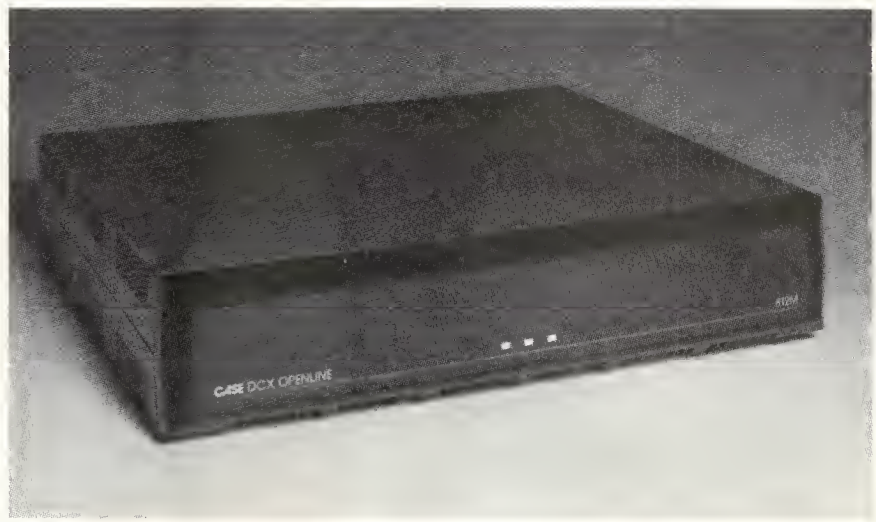
Concurrent Computer Corporation has added the XF/300 super-microcomputer to its line of compatible systems for the Unix operating system market.

The supermicro uses the Micro-XELOS operating system derived from Unix System V, Release 2, and features a dual-bus architecture with MC68000 processor and 256K-byte RAM chip. It provides 18 RS-232C ports, 1M byte to 4M bytes of memory, up to 177M bytes of formatted disk storage with combinations of 40M-byte and 59M-byte drives, 1.6M-byte floppy disk, and 17M-byte tape cartridge. The XF/300 offers optional hardware floating-point processor and IEEE 802.3 Ethernet data link controller.

The standard minimum configuration of the XF/300 is available for $14,000.

Concurrent Computer Corporation is located at 197 Hance Avenue, Tinton Falls, NJ 07724; (201) 758-7000.

Reader Service Number 15



The Case Communications DCX 812M is a four- or eight-port statistical multiplexer with a built-in, 9600/14400-bps speed Series 4000 modem. Trellis encoding is used at 14400 bps. The software-configurable 812M invokes diagnostic functions via a terminal attached to the supervisor port. List prices start at $3190. Contact Case Communications at 2120 Industrial Parkway, Silver Spring, MD 20904-1999; (301) 381-2300.

Reader Service Number 16

Dial-up modem operates at 9600 bps

Racal-Vadic is introducing the 9600VP dial-up modem, which operates at 9600 bps with error control and data compression and matches full-duplex asynchronous or half-duplex synchronous protocols.

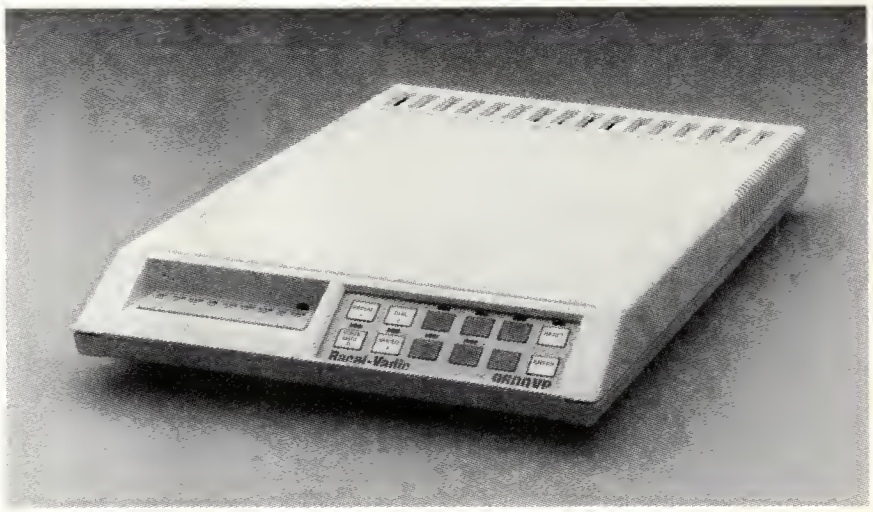
The Bell 103- and 212A-compatible 9600VP uses a CCITT modulation technique and the company's Data Link protocol at higher speeds. It offers automatic originate and answer capabilities, front-panel dialing and diagnostics, software-configurable options, and support for extended AT dialing protocol command sets.

The 9600VP uses MNP error control and correction protocol to eliminate data-transmission errors. Automatic speed conversion allows the 9600VP to receive data at a speed different from that of the communication link or of the receiving terminal. Fifteen telephone numbers can be stored in the nonvolatile memory; PBX systems and networks such as Sprint and MCI can be accessed.

The 9600VP is priced at $1495.

Racal-Vadic is headquartered at 1525 McCarthy Boulevard, Milpitas, CA 95035; (408) 946-2227.

Reader Service Number 17



Racal-Vadic's 9600VP full-duplex modem operates at 9600, 1200, and 300 bps.

IBM card includes 1200-baud modem

Omnipak from OmniTel, Inc., is a single-board multifunction card integrated with a Hayes-compatible 1200/300-bps modem and manufactured for use with the IBM PC, XT, AT, and compatibles. Omnipak comes packaged with communications and utilities software, such as RAM disk, printer spooler, and clock/calendar, enabling the personal computer to function as a communicating workstation.

Omnipak sells for $499 and includes a two-year warranty.

OmniTel, Inc., is located at 5415 Randall Place, Fremont, CA 94538; (415) 490-2202.

Reader Service Number 18

AT&T offers integrated voice/data workstation

AT&T's PC/PBX Connection is a line of hardware and software products designed to integrate personal computers and digital telephones. The workstations provide on-screen access to 200 voice and data features of the AT&T System 75 and System 85 digital PBX communications systems and simultaneous voice/data communications at 64K bps.

AT&T's PC/PBX products provide an element of multitasking to the single-user MS-DOS or PC-DOS operating systems by allowing a spreadsheet, file transfer, or phone call to be handled simultaneously.

Two basic hardware configurations of PC/PBX Connection exist: the 7404D PC cartridge for basic functions and the PC expansion card for full-function capability. The cartridge configuration supports data communications up to 9.6K bps. The full-function arrangement of PC expansion cards with software packages for MS-DOS or Unix operation systems will transfer data to other PCs or to hosts via AT&T's Digital Multiplexed Interface.

With an existing AT&T 7404D voice/data telephone and PC, the 7404D cartridge can be added for $175 and full functionality for $700.

Interested software vendors can contact AT&T's Information Systems Group at (201) 898-6710.

Reader Service Number 19

Device programs 5-volt PROMs

International Microsystems's EPROM-1 programs standard NMOS and CMOS EPROM/EEPROM devices up to 512K, without the need for additional hardware. The field-upgradable instrument is designed for engineering, field service, and small production runs.

The 64K-RAM EPROM-1 operates alone for the duplication of master devices through the use of Blank, Program, and Verify key entries. In addition, the $9.5 \times 6.2 \times 2$-inch programmer can support most standard computers through an RS-232C port; included with EPROM-1 is IBM PC editing software.

For more information on the $495 EPROM-1 contact International Microsystems, 11554 C Avenue, Auburn, CA 95603; (800) 325-6028 or (916) 885-7262.

Reader Service Number 20

Integrated business software available for LANs

A multiuser version of the Smart Software System is available to LAN users from Innovative Software. The integrated business applications contain networking versions of the Smart data manager, word processor, spreadsheet with graphics, communications and time manager, plus an 80,000-word spelling checker.

Version 3.0 is compatible with all DOS 3.10-compatible network operating systems such as Novell's Netware, 3 COM's 3+, the IBM PC network, and AT&T's Starlan. The system includes the Smart programming language, which permits complex applications development using procedural language and debugging commands. In addition, Smart's password protection schemes provide data integrity for all users of the network environment.

The suggested retail price for the complete Smart Software System LAN Multiuser Version 3.0 is $1795, which includes three workstations. Additional user-access nodes can be purchased for $595 each. Trade-up discounts are available.

Product information can be obtained by calling 800-GET-SMART, or in Kansas (913) 492-3800.

Reader Service Number 22

Standby power systems support PCs, disk drives

LorTec Power Systems has introduced three standby power systems with capacities of 200, 400, and 1000 voltamperes. According to the company, the systems can isolate microcomputer loads from blackouts, severe brownouts, and extreme line transients that can disrupt computer operations and damage computer circuits.

The SPS units assure 10 to 20 minutes of reserve power during which a computer operator can close files, save programs, and shut a system down without data loss. The units are designed to be compatible with the IBM PC, XT, and AT, Compaq Deskpro, Deskpro 286, Compaq Portable computers, and all models from Apple, AT&T, and Hewlett-Packard.

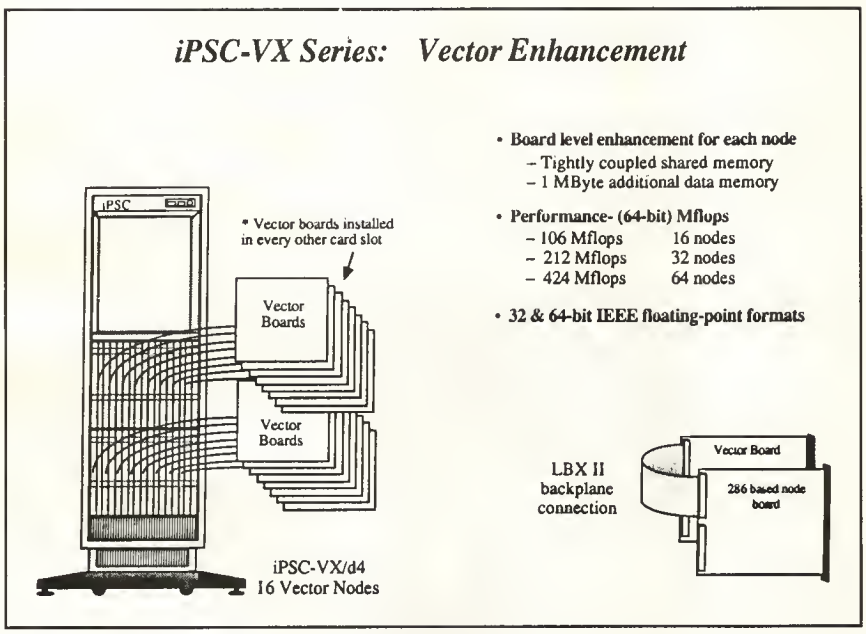
Prices for the SPS models begin at $495, depending on power capacity.

For more information contact LorTec Power Systems, 145 Keep Court, Elyria, OH 44035; (800) 222-2600, ext. 265.

Reader Service Number 21

Intel introduces vector processors

Intel Corp. is introducing vector processing versions of its iPSC family of concurrent computers. The iPSC-VX series consists of three upgradable models, the iPSC-VX/d6 64-node system offering 96M bytes of memory; the iPSC-VX/d5, a 32-node, 48M-byte version; and the iPSC-VX/d4, a 16-node, 24M-byte version. Software development tools and applications libraries are available for the series at introduction.



Intel's iPSC-VX series expands on the iPSC concurrent computer series. As can be seen here, a Multibus II iLBX bus allows each of the node processors to be tightly coupled to a vector processor, whose control code is loaded directly from the node processor and contains a user-accessible library of vector, scalar, and logical operations.

The company states that the /d6 delivers up to 424 MFLOPS at 64-bit arithmetic precision. It is designed to meet the performance requirements of computationally intensive applications including circuit simulation, computational chemistry, fluid dynamics, and oil-reservoir modeling.

Parallel numerical computation packages called LINcube and EIScube provide application libraries for scientific uses. Fortran development capabilities with simulation and debugging facilities in a Xenix operation environment are included for program development.

The iPSC-VX/d6 64-processor system sells for $850,000, the /d5 with 32 nodes costs $450,000, and the /d4 with 16 nodes is priced at $250,000. Prices for iPSC upgrades to the iPSC-VX series are available on request.

More information is available from Intel Scientific Computers, Customer Response Dept., 15201 NW Greenbrier Parkway, Beaverton, OR 97006; (503) 629-7629.

Reader Service Number 23

Full-duplex, 9600/4800-bps modem supports CCITT V.32

Concord Data Systems announces its CDS V.32 Trellis, a full-duplex, dial/lease-line modem that supports the CCITT V.32 recommendation.

Available in domestic and international models, the modem operates at 9600/4800-bps speeds over dial-lines and over 4-wire/2-wire leased lines for point-to-point communications. Leased-line users can switch the modem to dial-line transmissions as needed. Dial users connect with the Public Switched Telephone Network and can transmit at higher speeds without converting to leased-line installations.

The modem offers echo cancelling and a forward error correction technique called convolutional encoding, or trellis coding. Trellis coding reduces errors in transmitted data caused by channel noise or other signal distortion and adds a redundancy factor to the transmitted signal to recognize invalid signal patterns.

The CDS modem sells for $3495. Quantity discounts are available.

For more information contact Concord Data Systems, 397 Williams Street, Marlborough, MA 01752; (617) 460-0808.

Reader Service Number 24

The Concord Data Systems V.32 Trellis modem is powered by CSA/VDE/FCC/UL-approved integral power supply; it uses convection cooling to control operating temperatures.

Modem fits IBM short slots

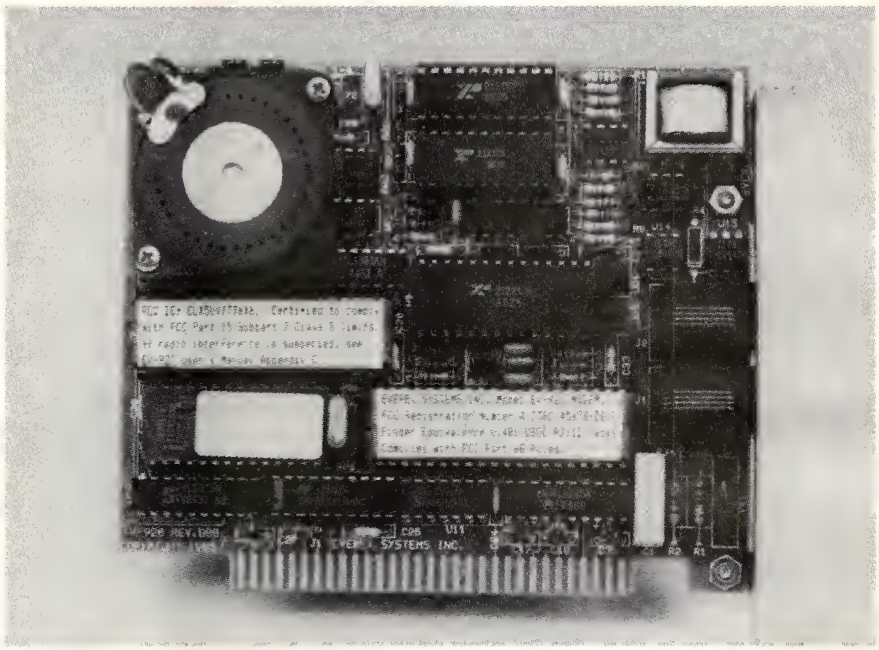
A 300/1200-baud modem on a half-size card has been designed by Everex Systems to fit the short slots in IBM XTs, Portables, and compatibles. The Hayes-compatible Evercom Half Card features call-progress monitoring, voice and data communications with automatic transition, audio and data telephone-line monitoring, half- and full-duplex data transmission, and a built-in loudspeaker under software volume control.

The Evercom is supplied with BIT COM software that supports 132-column displays, includes built-in data buffers, and enables other programs to be run while remaining available to accept more data communications.

Suggested retail price of the Evercom Half Card is $249.

For further information contact Everex Systems, 47777 Warm Springs Boulevard, Fremont, CA 94539; (415) 498-1111.

Reader Service Number 25



The Everex Systems Evercom Half-Card modem can store up to 32,000 phone numbers in its personal dialing directory.

ECL/TTL gate array added to Honeywell line

Honeywell has announced a low-power ECL/TTL gate array that functions within military temperature requirements.

The HM3500 uses a 2.5-micron, oxide-isolated, digital bipolar process. The gate array operates at system clock frequencies of more than 300 MHz and dissipates 3.5 watts. Typical internal CML gate delays are 400 picoseconds. A 148-pin grid array is the standard package for commercial applications in mainframe and super-minicomputers, automatic test equipment, and high-speed telecommunications.

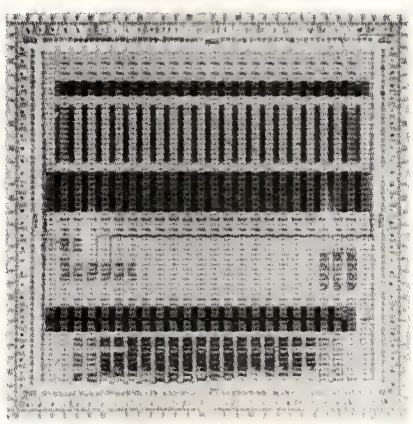
Nonrecurring engineering charges for the HM3500 in commercial applications are approximately $60,000, which includes 10 working samples. Typical turnaround time is 12 weeks.

For further information call (800) 328-5111, ext. 3422, or write Semi-custom Product Marketing, Honeywell Digital Product Center, 1150 East Cheyenne Mountain Boulevard, Colorado Springs, CO 80906.

Reader Service Number 26

ASIC offers 8000 gates

Honeywell has introduced a 1.25-micron, bipolar application-specific integrated circuit designed for supercomputers, super-minicomputers, automatic test equipment, communications, and graphics.



Honeywell's dense 1.25-micron, bipolar ASIC, the HE8000, is available in a 235-pin, pin-grid-array package. The die, mounted on a copper/tungsten alloy, provides a low thermal resistance path to the heat sink.

The HE8000 is an 8000-gate array featuring current mode logic and macrocells that achieve 6 to 8 watts of dissipation while operating at 210 picoseconds. According to the company, on-chip testability, programmable-drive macrocells, and computer-aided design support enable designers to increase system density, performance, and reliability.

Pricing was not announced. For information on the HE8000 gate array ASIC, contact Honeywell Bipolar Product Marketing, Digital Product Center, 1150 East Cheyenne Mountain Boulevard, Colorado Springs, CO 80906; (800) 328-5111, ext. 3422.

Reader Service Number 27

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Interest Card.

High 192 Medium 193 Low 194

Product Summary

Editor: Kenneth Majithia/IBM Corporation

For more information, circle the appropriate Reader Service Number on the Reader Service Card at the back of the magazine.

MANUFACTURER	MODEL	COMMENTS	RS NO.
Chips/Components			
Advanced Micro Devices PO Box 3453 Sunnyvale, CA 94088 (408) 732-2400	Am27C1024 EPROM	One-megabit CMOS EPROM for 16- and 32-bit systems has a 200-ns access time, an under 2-minute programming time (using AMD's interactive algorithm), and an organization of 64K words by 16 bits. A silicon signature feature identifies the part type and manufacturer in automatic programming machines. Priced at $199 each in 100-unit quantities.	60
	Am29X305A microcontroller	Eight-bit microcontroller functions as an alternate source, plug-in replacement for the 200-ns cycle time Signetics 8X305. Features on-chip registers and data-handling capability. $29.70 each for 50-pin side brazed ceramic DIP packages in 100-unit quantities.	61
	Am7996 transceiver	Integrates network transceiver functions, including transmit, receive, and collision detection. When combined with the Am7990 LAN controller for Ethernet (Lance) and the Am7992B serial interface adapter, provides integrated solution for IEEE 802.3, Ethernet, and Cheapernet networking applications. $23.90 each for 20-pin ceramic DIPs in 100-unit quantities.	62
National Semiconductor PO Box 58090 Santa Clara, CA 95052-8090 (408) 721-5000	HPC16040 microcontroller	The 240-ns, 2-micron CMOS chip contains a 16-bit CPU with 4K bytes of ROM, 256 bytes of RAM, 52 general-purpose I/O lines, eight 16-bit timers, and UART for data communications. Packaged in a 68-pin plastic leadless chip carrier. Sample quantity pricing is $29.90 each.	63
NCR Corporation Dayton, OH 45479 (513) 445-3467	53C80 controller	CMOS Small Computer System interface drives the SCSI bus with on-chip, single-ended bus transceivers. Communicates with the microprocessor as a peripheral device and is controlled by reading and writing internal registers, which can be addressed as standard or memory-mapped I/O. Surface-mountable, 44-pin plastic chip carriers or 48-pin DIPs. $13.65 each in quantities of 1000.	64
Standard Microsystems 35 Marcus Boulevard Hauppauge, NY 11788 (516) 273-3100	HYC9058 LAN transceiver	High-impedance transceiver contains drive circuitry to guarantee noise immunity for data transfer over a local area network line; can be incorporated into existing Arcnet networks. Assembled in a 20-pin, single in-line package and offered in either straight or bent (90-degree right angle) lead frame. Priced at $32.35 in 100-piece quantity.	65
VLSI Technology 10220 South 51st Street Phoenix, AZ 85044 (602) 893-8574	VL82C389 MPC	The 149-pin message-passing coprocessor for Multibus II-based systems controls solicted messages (DMA transfers used to move blocks of data within the system) and unsolicited messages (intelligent interrupts). The monolithic IC with 70,000 transistors is implemented in 2-micron, double-metal CMOS. Sample quantity pricing is $200.	66
Boards			
Anchor Automation 6913 Valjean Avenue Van Nuys, CA 91406 (714) 995-1650	Signalman LIGHTNING	Auto-dial, 2400-bps, add-in half-card modem for IBM PCs and compatibles operates asynchronously with an automatic equalizer and uses the Hayes-compatible modem command structure. Can be bundled with LYNC communication software from Norton-Lambert. $499; dealer and OEM discounts available.	67

Manufacturer	Model	Comments	Rs No.
Densan Co. 17982 Skypark Circle Suite B Irvine, CA 92714 (714) 261-5220	DSB-020PL RAM	Two-megabyte RAM provides zero-wait-state operation when combined with Multibus CPU boards, containing either the Intel high-speed synchronous interface or the asynchronous iLBX memory expansion interface. Features 187-ns maximum access time. $971 in quantities of 100.	68

Communications

Able Computer 3080 Airway Avenue Costa Mesa, CA 92626 (714) 979-7030	Muxmaster multiplexer	Distributed multiplexing system connects users to MicroVAX II, VAX, and PDP-11 computers through twisted-pair cable capable of extending 2000 feet from the host. Software-compatible with VMS, MicroVMS, RSX, and RSTS. Supports full European-style modem control. Pricing begins at $3000 for the first 16-line cluster system.	69
US Robotics 8100 North McCormic Boulevard Skokie, IL 60076 (312) 982-5010	Rackmount 30 modem	Modular modem system provides dial-up 2400/1200/300-bps, asynchronous data communications for mainframe, minicomputers, and multiuser microcomputers. Includes 19-inch rack chassis, 25 dual modem boards, 15 dual interface boards, two power supply units, an AC and a DC power distribution board, and a protective front panel. Pricing depends on components ordered.	70
Valid Logic Systems 2820 Orchard Parkway San Jose, CA 95134 (408) 945-9400	Networked Realchip	Compatible with IBM, DEC, and Valid workstations via Ethernet, this application-specific hardware provides designers who are validating their electronic designs with concurrent, multiuser access to models of VLSI devices. Consists of Realchip hardware, a network server processor, and an Ethernet interface. Basic configuration is priced at $49,500.	71

Peripherals

Ava Instrumentation 8010 Highway 9 Ben Lomond, CA 95005 (408) 336-2281	409 drive tester	Floppy-disk drive tester is designed to work with high-density floppies such as the 1.6M-byte, dual-speed and Kodak 3.3M-byte and 6.6M-byte floppy drives, in addition to 8-, 5.25-, and 3.5-inch drives. A nonvolatile RAM stores drive parameters and test programs with the power off. $3200. Quantity discounts available.	72
Intel Corporation 5200 N.E. Elam Young Parkway Hillsboro, OR 97123 (503) 681-2279	P-MON386ES monitor	ES version of PSCOPE debugging monitor for 32-bit, 80386-based systems includes host software that runs on the company's 286/310 system with Xenix 3.0 operating system. Users can download programs into the target prototype memory, set hardware/software breakpoints, and examine and modify memory and processor registers. $3495.	73
Tektronix PO Box 500 Beaverton, OR 97077 (503) 644-0161	CX4111 terminal	Computer terminal displays 1024 × 768 pixels with a virtual graphics space of four billion by four billion points. Sixteen colors from a palette of 4096 shades can be displayed simultaneously on the 19-inch screen. Priced at $13,950 (US only).	74

Software

Abacus Software PO Box 7211 Grand Rapids, MI 49510 (616) 241-5510	PCBoard Designer	Atari ST CAD tool automates printed circuit board layouts and features automatic routing capability. Layouts can be printed on Epson or compatible dot matrix printers. $395.	75
Communications Research Group 8939 Jefferson Highway Baton Rouge, LA 70809 (504) 923-0888	Unix Blast software	Package connects the Unix PC with 120 computers under 25 major operating systems, allowing the Unix PC to transfer files to and from any other computer running Blast software. According to the company, no add-on boards or interface hardware are needed, and RS-232 serial ports provide full connection on both ends. $295.	76

Access

Recent articles on microcomputing

Editor: Peter R. Rony/Department of Chemical Engineering/University of Delaware/Newark, DE 19716

Micro/Systems Journal

Vol. 1, No. 1, Mar./Apr. 1985:

S. Kolandsky, "Bringing Up CP/M-Plus," pp. 20-28.

R. Kreymborg, "Assembly Language Extensions for MS-Basic," pp. 36-42.

D. Brewer, "New Tricks for CP/M-2.2," pp. 46-55.

W. Wong, "16-Bit Lisp and Prolog," pp. 62-66.

Vol. 1, No. 2, May/June 1985:

J. Monahan, "Build an S-100 to PC-Bus Converter," pp. 24-30.

W. Wong, "Interfacing to MS-DOS," pp. 32-35.

E. Heyman, "C and the Godbout Disk 1 Controller," pp. 46-53.

W. G. Wong, "16-Bit Lisp and Prolog Implementations—Part II," pp. 56-63.

C. Sondgeroth, "Loadable BIOS Drivers for CP/M," pp. 66-71.

Vol. 1, No. 3, July/Aug. 1985:

D. N. Quinn, "Structured Programming With the Microsoft M80 Assembler," pp. 26-36.

S. Bosak, "Scientific and Technical Word Processors—Part I," pp. 40-46.

Vol. 1, No. 4, Sept./Oct 1985:

S. R. Davis, "Interrupt Borrowing With Turbo Pascal," pp. 34-40.

W. G. Wong, "Program Interfacing to MS-DOS—Part III," pp. 54-59.

A. G. W. Cameron, "Typesetting With or Without a Typesetter," pp. 64-72.

Vol. 1, No. 5, Nov./Dec. 1985:

S. R. Davis, "Turbocharge Your 8086/8088 Computer," pp. 32-43.

N. T. Carnevale, "Faster Floating-Point Math," pp. 46-54.

J. L. Calaway and B. Hill, Jr., "Bringing Up CP/M-68K," pp. 58-62.

W. G. Wong, "Program Interfacing to MS-DOS—Part IV," pp. 70-73.

Mini-Micro Systems

Vol. 18, No. 14, Nov. 1985:

L. Anderson and D. Welsh, "Application Generator Speeds Development," pp. 93-101.

J. Borrell, "Users Get More for Less in Graphics Terminals," pp. 107-115.

M. Tucker, "ASCII Software Claims New Markets," pp. 119-124.

Vol. 18, No. 15, Nov. 15, 1985:

Issue includes features on disk controllers, intelligent subsystems, and disk buffers as well as product guides on rigid disk drives and disk drive subsystems, matrix character printers, solid font character printers, and alphanumeric display terminals.

Vol. 18, No. 16, Dec. 1985:

C. Warren, "Optical Storage Shines on the Horizon," pp. 68-80.

J. Borrell, "Electronic Publishing Lands on the Desktop," pp. 85-92.

Vol. 19, No. 1, Jan. 1986:

M. Tucker, "32-Bit Chips Reshape Unix Industry," pp. 69-78.

D. De Salvo, "PC-Unix Links Penetrate Office," pp. 83-89.

J. Victor, "Micros Gain Sight With Image Boards," pp. 92-98.

J. Borrell, "Thermal Printing Heats Up Color Hard-Copy Market," pp. 103-107.

S. M. Pytka, "Copier/Printer Market to Explode by 1990," pp. 111-115.

Vol. 19, No. 2, Feb. 1986:

M. Tucker, "Tools Speed Software Integration," pp. 55-63.

C. Warren, "SCSI Opens Integration Opportunities," pp. 71-79.

1. D. Allan, "ESDI Joins Interface Ranks," pp. 83-91.

J. Fisher, "Super SCSI Controllers Boost I/O Performance," pp. 95-100.

Vol. 19, No. 3, Feb. 14, 1986:

Issue includes features on printer servers and LAN software, and product guides on voice-grade modems, local-area networks, and networking software.

Mosiac

Vol. 16, No. 4, 1985:

Special issue on advanced scientific computing. Articles include "Biology's Computational Future," "Computers to Go With the Flow," "Many Chemists, Many Cups of Tea," "Expert Chemical Systems," "Mathematicians at the Receiving End," and "Physicists: More Problems Than Approaches."

Mundo Electronico

No. 154, Oct. 1985:

M. Puigbo and M. Martinez, "Diseno de Equipos Para la Ensenanza Utilizando μP," pp. 85-91.

E. Mandado and J. Bernardez, "Procesadores Microprogramables, Unidades de Control," pp. 129-133.

J. J. Padilla Navidad, "El Sistema Operativo Unix," pp. 135-141.

No. 155, Nov. 1985:

F. Serra, J. Aguilo, J. Buxo, and M. Blafeur, "Technologia CMOS: Diseno de un Contador Binario de Four Bits," pp. 51-54.

C. N. Alfonso and C. Ortiz, "Procesado de Materiales por Laser: Su Aplicacion en Microelectronica y Optoelectronica," pp. 57-61.

F. Vidondo Morras, "Arquitecturas de Sistemas en Tiempo Real con Galileo," pp. 115-125.

No. 156, Dec. 1985:

R. Aparicio Lopez, "Metodo de Compensacion de la Potencia Reactiva por Computador," pp. 81-85.

J. M. Guell, W. Warzanskyj, and A. Castillo, "Diseno y Simulacion de Filtros Digitales FIR," pp. 103-109.

No. 156, Jan. 1986:

G. Nausmann, "El Unix y la Familia NS32000," pp. 78-82.

E. R. Veiga, "Circuitos Tridemensionales: La Alternativa Vertical al Diseno VLSI," pp. 95-100.

B. Dance, "Logica CMOS de Alta Velocidad (HCMOS)," pp. 103-107.

PC Magazine

Vol. 5, No. 4, Feb. 25, 1986:

C. Petzold, J. Duntemann, P. Chisholm, and M. D. Stone, "Operating in a New Environment," pp. 108-132.

F. J. Derfler, Jr., J. Pepper, J. Taylor, and W. L. Roch, "Hardware: Computing Options for Power Users," pp. 134-161.

G. Hart, "Building a Better Mouse Interface," pp. 167-172.

L. L. Meadows, "The Returns Are In: Tax Software for 1985," pp. 217-230.

Vol. 5, No. 5, Mar. 11, 1986:

G. Hart, "CAD: The Big Picture for Micros," pp. 106-164.

G. Hart, "CAD Support: An Embarrassment of Riches," pp. 169-180.

F. J. Derfler, Jr. and W. Stallings, "The IBM Token-Ring LAN: What? Why? Now?" pp. 197-206.

S. R. Davis, "Getting the Most Out of Your Hard Disk," pp. 227-233.

Vol. 5, No. 6, Mar. 25, 1986:

C. Petzold, "The EGA Standard: Monitors That Measure Up," pp. 108-120.

J. Forney, V. Puglia, and P. Wiswell, "Color Monitors: Challenging the IBM Standard," pp. 123-144.

S. Stallings, "AT&T's Wonder Boards," pp. 151-171.

C. Bermant, "Hard Disk Cards: An Expensive Solution Worth Considering," pp. 185-196.

J. Dickinson, "The Business of Words: Outliners," pp. 199-220.

Science

Vol. 231, No. 4741, Feb. 28, 1986:

Special issue on computers. Articles include:

J. P. Crecine, "The Next Generation of Personal Computers," pp. 935-943.

D. M. Jennings et al., "Computer Networking for Scientists," pp. 943-957.

R. Davis, "Knowledge-Based Systems," pp. 957-963.

F. Baskett and J. L. Hennessy, "Small Shared-Memory Multiprocessors," pp. 963-967.

D. J. Kuck, E. S. Davidson, D. H. Lawrie, and A. H. Sameh, "Parallel Supercomputing Today and the Cedar Approach." pp. 967-974.

R. P. Gabriel, "Massively Parallel Computers: The Connection Machine and Non-Von," pp. 975-978.

Addresses of Publishers

A+
11 Davis Dr.
Belmont, CA 94002
(415) 598-2290

Addison Wesley Publishing Co., Inc.
Jacob Way
Reading, MA 01867
(617) 944-3700

Business Computer Systems
Cahners Publishing Co.
270 St. Paul St.
Denver, CO 80206
(303) 388-4511

Byte
Byte Publications Inc.
70 Main St.
Peterborough, NH 03458
(603) 924-9281

Chapman and Hall/Methuen Inc.
733 Third Ave.
New York, NY 10017
(212) 922-3550

Computer
IEEE Computer Society
10662 Los Vaqueros Cir.
Los Alamitos, CA 90720
(714) 821-8380

Datamation
Technical Publishing Co.
1301 S. Grove Ave.
Barrington, IL 60010
(312) 774-8115

Dr. Dobb's Journal
People's Computer Co.
2464 Embarcadero Way
Palo Alto, CA 94304
(415) 424-0600

Electronic Design
50 Essex St.
Rochelle Park, NJ 07662
(201) 843-0550

Electronics
McGraw-Hill Publications Co.
1221 Ave. of the Americas
New York, NY 10020
(212) 512-2000

Hewlett-Packard Journal
Hewlett-Packard Co.
3000 Hanover St.
Palo Alto, CA 94304
(415) 857-3853

IEEE Transactions (all)
IEEE Service Center
445 Hoes Lane
Piscataway, NJ 08854
(201) 981-0060, x133

Microcomputer Applications
International Society for Mini and Microcomputers
Acta Press
Box 2481
Anaheim, CA 92804

Microprocessors and Microsystems
Business Press International
205 E. 42nd St., Suite 1705
New York, NY 10017
(212) 867-2080

or

Butterworth Scientific Ltd.
Journals Division
PO Box 63
Westbury House
Bury St.
Guildford, Surrey GU2 5BH UK

Micro/Systems Journal
Libes, Inc.
995 Chimney Ridge
Springfield, NJ 07081

Mini-Micro Systems
Cahners Publishing Co.
270 St. Paul St.
Denver, CO 80206
(303) 388-4511

MIT Press
28 Carleton St.
Cambridge, MA 02142
(617) 253-2884

Mitchell Publishing, Inc.
915 River St.
Santa Cruz, CA 95060
(408) 425-3851

Mosaic
National Science Foundation
1800 G St. NW
Washington, DC 20550
(202) 655-4000

Mundo Electronico
Boixareau Editores, SA
Grand Via de les Corts Catalanes 594
2 Barcelona 7 Spain

PC Magazine
Ziff-Davis Publishing Co.
One Park Ave.
New York, NY 10016
(212) 503-3500

Prentice-Hall Inc.
200 Old Tappan Rd.
Old Tappan, NJ 07675
(201) 767-5053

Science
American Association for the Advancement of Science
1333 H St. NW
Washington, DC 20005

Software Masters
PO Box 3638
Bryan, TX 77805

Advertiser Index—June 1986

FOR DISPLAY ADVERTISING INFORMATION CONTACT

Southern California and Mountain States: Richard C. Faust Company, 24050 Madison Street, Suite 100, Torrance, CA 90505; (213) 373-9604.
Northern California and Pacific Northwest: Don Farris Company, 161 W. 25th Ave., #102B, San Mateo, CA 94403; (415) 349-2222.
Jack Vance, P.O. Box 3205, Saratoga, CA 95070; (408) 741-0354.
East Coast: Hart Associates, Inc., P.O. Box 339, 42 Lake Blvd., Matawan, NJ 07747; (201) 583-8500.
New England: Arpin Associates, P.O. Box 227, Weston, MA 02193; (617) 899-5613.
George Watts, III, 4 Conifer Dr., Wilbraham, MA 01095; (413) 596-4747.
Midwest: Thomas Knorr, Knorr MicroMedia, Inc., 333 North Michigan Ave., Chicago, IL 60601; (312) 726-2633.
Southwest: The House Company, 3000 Weslayan, Suite 345, Houston, TX 77027; (713) 622-2868.

Advertising Director: Mike Koehler, *IEEE MICRO*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720; (714) 821-3240, 821-8380.

For production information, conference or classified advertising contact Carole L. Porter, *IEEE MICRO*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720; (714) 821-1140.

Product Index

IEEE MICRO

For further information on advertised products, new products, or literature, fill out the **Reader Service Card** (top). Circle the number on the RS Card that corresponds to the number of the item for which you would like more information.

To indicate your interest in an article or department, fill out the **Reader Interest Card** (bottom). Circle the number on the RI Card that corresponds to the level of interest given in the Reader Interest Survey at the end of the article or department.

Please print or type your name and address.

READER SERVICE CARD

IEEE MICRO INFORMATION ABOUT PRODUCTS

Void after December 31, 1986 **6/86**

Name _____

Company _____

Address _____

City _____

State _____

Zip _____

Country _____

Title _____

Telephone number ( ) _____

Product announcements, and products for review, should be sent to Marie English, Managing Editor, *IEEE Micro*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578.

PRODUCTS PURCHASED OR SPECIFIED	FOR JOB	FOR HOBBY
Computers	80	81
Peripherals	82	83
Data communications equip.	84	85
Memories, components	86	87
Software and services	88	89
Publications	90	91
Other	92	93

94 Please send me information on advertising in *IEEE Micro*.

95 Please send me the IEEE-CS *Publications Catalog*.

96 Please send me an IEEE Fellow nomination form.

97 Please send me an IEEE Computer Society membership application.

98 Please send me an *IEEE Micro* author's guide.

Send more information on numbered items:

1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61	65	69	73	77	81	85	89	93	97
2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62	66	70	74	78	82	86	90	94	98
3	7	11	15	19	23	27	31	35	39	43	47	51	55	59	63	67	71	75	79	83	87	91	95	99
4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68	72	76	80	84	88	92	96	100

READER INTEREST CARD

IEEE MICRO EDITORIAL RESPONSE

6/86

Name _____

Company _____

Address _____

City _____

State _____

Zip _____

Country _____

Title _____

Telephone number ( ) _____

Comments:

Reader Interest Survey:

101	116	131	146	161	176	191
102	117	132	147	162	177	192
103	118	133	148	163	178	193
104	119	134	149	164	179	194
105	120	135	150	165	180	195
106	121	136	151	166	181	196
107	122	137	152	167	182	197
108	123	138	153	168	183	198
109	124	139	154	169	184	199
110	125	140	155	170	185	200
111	126	141	156	171	186	201
112	127	142	157	172	187	202
113	128	143	158	173	188	203
114	129	144	159	174	189	204
115	130	145	160	175	190	205

Continue comments on other side

This PO box for reader service cards only.

PLACE STAMP HERE

IEEE MICRO

Reader Service Inquiries
Box 24168
Los Angeles, CA 90024
USA

Comments on articles and other editorial matter:

I liked _____

I disliked _____

I would like _____

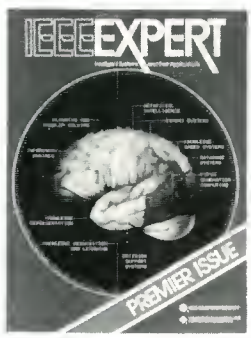
For Reader Interest Survey, see other side

PLACE STAMP HERE

IEEE MICRO

10662 Los Vaqueros Circle
Los Alamitos, CA 90720-2578
USA

We Set The Standards



IEEE COMPUTER SOCIETY

Standards define a product and create its market.

Take data communications, for example. This multibillion-dollar industry depends on standards to make competing product lines work together. Who leads the effort to set these standards? The IEEE Computer Society.

The computer industry's most influential managers, engineers, and scientists belong to the IEEE Computer Society and subscribe to our publications. They hold advanced degrees, play key roles in large corporations, and control computer equipment purchases.

These professionals rely upon IEEE Computer Society publications to keep up with evolving industry standards. They also turn to us for insights into the technology and its applications—insights that affect the way they do business.

Computer Society publications uphold the standards of objectivity, depth, and timeliness that professionals need to compete effectively in the marketplace. That's why our subscribers find us a valued investment.

Whether you sell equipment or services, you pay attention to standards. Doesn't it make sense to go directly to the source?

The IEEE Computer Society publications: We set the standards.

Regional Sales Representatives

So. Cal. & Mtn. States
Richard C. Faust
24050 Madison St.,#100
Torrance, CA 90505
(213) 373-9604

No. Cal. & Pacific Northwest
Don Farris, Jack Vance
161 West 25th Avenue #102B
San Mateo, CA 94403
(415) 349-2222

Southwest
Royce House, Roy Nelson
3817 Richmond Ave., #110
Houston, TX 77027
(713) 622-2868

Midwest
Thomas Knorr
Knorr MicroMedia, Inc.
333 North Michigan Ave.
Chicago, IL 60601
(312) 726-2633

New England
Lee Arpin
P.O. Box 227
Weston, MA 02193
(617) 899-5613

George Watts, III
4 Conifer Drive
Wilbraham, MA 01095
(413) 596-4747

East Coast
Hart Associates
P.O. Box 339
42 Lake Blvd.
Matawan, N.J. 07747
(201) 583-8500

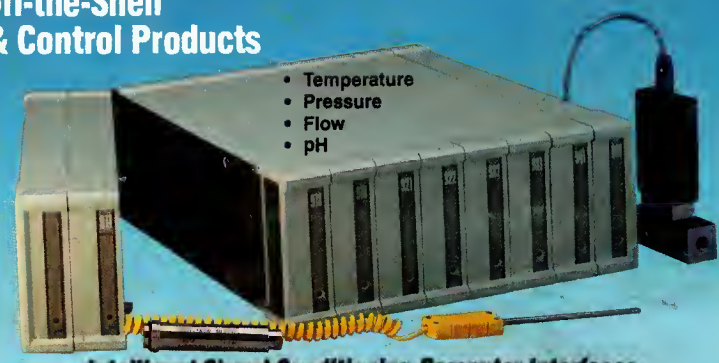
Mike Koehler, Advertising Director • IEEE Computer Society • 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578 • (714) 821-8380

GET THE NEW! FREE! 1986 HANDBOOKS *NOW!*

✓ Thousands of Off-the-Shelf Measurement & Control Products

Precision X-Y and Y-t Recorders

- Temperature
- Pressure
- Flow
- pH



Intelligent Signal Conditioning Computer Interfaces

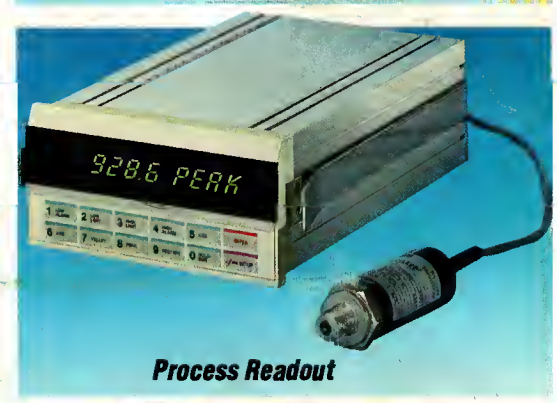
pH Instrumentation

Totalize Flow

Data Log

Strain Gage Amplifier

Pressure Transducers



Process Readout

- *Everything at Your Fingertips!*
- *Off-the-Shelf for Immediate Delivery!*
- *All Prices Included In Every Handbook!*
- *Largest Selection of Temperature, Pressure, Flow & pH Products in the World!*

For placing orders, dial (203) 359-1660!

OMEGA ENGINEERING, INC.

(203) 359-RUSH

One Omega Drive, Box 4047, Stamford, CT 06907
Telex 996404 Cable OMEGA FAX (203) 359-7700

Circle Reader Service Number to Receive Your NEW! FREE! Handbooks.